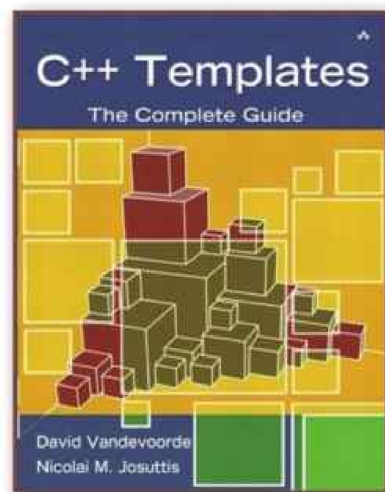
PEARSON

C和C++实务精选
品味岁月积淀，读享技术菁华

C++ Templates

中文版

[美] David Vandevoorde [德]Nicolai M. Josuttis 著
陈伟柱 译



C++ Templates: The Complete Guide

- 详细讲解C++模板的概念与使用技巧
- 透彻剖析C++模板特性的强大功能
- *C/C++ Users Journal*前任总编Marc Briand倾力推荐



人民邮电出版社
POSTS & TELECOM PRESS





PEARSON

C++Templates[illegible]

[[illegible]]David Vandevoorde [[illegible]]Nicolai M.Josuttis [illegible]

[illegible] [illegible]

[illegible]

[illegible]

CIP

C++Templates/Vandevoorde,D.Josuttis,N.M..--2013.4

ISBN 978-7-115-31281-5

Ⅰ.①C… Ⅱ.①…②…③… Ⅲ.①C— Ⅳ.①TP312

CIP2013045000

David Vandevoorde,Nicolai M.Josuttis:C++ Templates: The Complete Guide

ISBN: 0201734842

C++ Templates

◆ []David Vandevoorde []Nicolai M.Josuttis

◆ 14

□□ 100061 □□□□ 315@ptpress.com.cn

□□ http://www.ptpress.com.cn

□□□□□□□□□□□□□

◆□□□800×1000 1/16

□□□32.25

□□□716□□ 2013□4□□1□

□□□1-3000□ 2013□4□□□1□□□

□□□□□□□□ □□□01-2013-1020□

ISBN 978-7-115-31281-5

□□□89.00□

□□□□□□□(010)67132692 □□□□□□□(010)67129223

□□□□□□(010)67171154

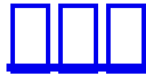
□□□□□□□□□□□□□□□0021□

[illegible]

[illegible]C++[illegible]C++[illegible]

[illegible]22[illegible]1[illegible]1[illegible]2[illegible]7[illegible]2[illegible]8[illegible]13[illegible]3[illegible]14[illegible]18[illegible]C++[illegible]4[illegible]19[illegible]22[illegible]A[illegible]B[illegible]

[illegible]C++[illegible]C++[illegible]



C++[illegible] C++[illegible]C++[illegible]“[illegible]C++”[illegible]“[illegible]C++”[illegible]C++[illegible]“[illegible]”[illegible]C++[illegible]

[illegible]C++[illegible]GP[illegible]boost[illegible]GP[illegible]C++[illegible]GP[illegible]C++[illegible]GP[illegible]OO[illegible]

[illegible]

[illegible]3[illegible]1[illegible]

C++[illegible] C++[illegible]5[illegible]

[illegible]“[illegible]”[illegible]

[illegible]

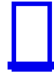
[illegible]

[illegible]1[illegible]16[illegible]C++[illegible]

[illegible]

[illegible]

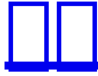
2003[illegible]12[illegible]



C++ Template 1990 Annotated C++Reference Manual “ARM” [EllisStroustrupARM] C++

David C++ Nico “ ” C++ C++

“ ” policy metaprogramming



[illegible]

[illegible] Kyle Blaney [illegible] Thomas Gschwind [illegible] Dennis Mancl [illegible] Patrick Mc Killen [illegible] Jan Christiaan Van Winkel [illegible] Dietmar Kühl [illegible]

[illegible] Edison Design Group [illegible] C++ [illegible] GNU [illegible] egcs [illegible] Jason Merrill [illegible] Microsoft [illegible] Visual C++ [illegible] Jonathan Caves [illegible] Herb Sutter [illegible] Jason Shirk [illegible]

[illegible] “C++ Wisdom” [illegible] C++ [illegible] comp.lang.c++.moderated [illegible] comp.std.c++ [illegible]

Addison-Wesley [illegible] Debbie Lafferty [illegible] Tyrrell Albaugh [illegible] Bunny Ames [illegible] Melanie Buck [illegible] Jacquelyn Doucette [illegible] Chanda Leary-Coutu [illegible] Catherine Ohala [illegible] Marty Rabinowitz [illegible] Marina Lang [illegible] Addison-Wesley [illegible] Susan Winer [illegible]

Nico[illegible]

[illegible]Ulli[illegible]Lucas[illegible]Anica[illegible]Frederic[illegible]

[illegible] David[illegible]

David[illegible]

[illegible]Karina[illegible]“[illegible]”[illegible]Karina [illegible]“[illegible]”[illegible]

[illegible]Nico[illegible]Nico[illegible]

John“Mr.Template”Spicer [illegible] Steve“Mr.Overload”Adamczyk [illegible]C++[illegible]C++[illegible]element[illegible]

[illegible]“[illegible]”[illegible]Michael Beckmann[illegible]Brett and Julie Beene[illegible]Jarran Carr[illegible]Simon Chang[illegible]Ho and Sarah Cho[illegible]Christophe De Dinechin[illegible]Ewa Deelman[illegible]Neil Eberle[illegible]Sassan Hazeghi[illegible]Vikram Kumar[illegible]Jim[illegible]Lindsay Long[illegible]R.J.Morgan[illegible]Mike Puritano[illegible]Ragu Raghavendra[illegible]Jim[illegible]Phuong Sharp[illegible]Gregg Vaughn[illegible]John Wiegley[illegible]

[illegible]1[illegible] [illegible]

[illegible] C++[illegible] C++[illegible]paradigm[illegible]

[illegible]C++[illegible]C++[illegible]
C++[illegible]

[illegible]

•David[illegible]C++[illegible]

•Nico[illegible]

[illegible]—[illegible]

Addison-Wesley[illegible]
C++[illegible]C++[illegible]

1.1 [illegible]

[illegible]C++[illegible]IOstream[illegible]C++[illegible]C++[illegible]

[illegible]1998[illegible]C++[illegible][Standard98][illegible]C++[illegible][Standard02][illegible]C++[illegible][SroustrupC++PL][illegible][JosuttisOOP][illegible][JosuttisStdLib][illegible]C++[illegible]B.3.5[illegible]

1.2 [illegible]

[illegible]

- [illegible]1[illegible]
- [illegible]2[illegible]
- [illegible]3[illegible]C++[illegible]
- [illegible]4[illegible]

[illegible]C++[illegible]

[illegible]1[illegible]3[illegible]2[illegible]22[illegible]20[illegible]

1.3 [illegible]

[illegible]C++[illegible]1[illegible]——[illegible]1[illegible]

[illegible]2[illegible]3[illegible]2[illegible]4[illegible]3[illegible]4[illegible]3[illegible]

[illegible]

[illegible]C++[illegible]http://www.josuttis.com/tmplbook/[illegible]

1.4 [illegible]

C++[illegible]

□□□□□□□□□□□□□□1□□4□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□“□□□□□□□□”□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
void foo (const int &x);
void foo (const int& x);
void foo (int const &x);
void foo (int const& x);
```

□“□□□”□□□□□□□□□□□□□□□□□□□□□□□□□int const□□□□□const int□□□□□□□□□□□□□□□□□□□□□□□□“□□□□□□□□”□int const□□□□□□□□□□□□□□□□“□□□□□□”□□□const□□□□□□□□□□□□□□

```
const int N = 100;
```

□□□□

```
int const N = 100;
```

□□□□□

```
int* const book mark;    //□□□□□□□□□□□□□□□□□□
```

□□□□□□□□□□□□□□□□□const□□□□□□□□ * □□□ int □□□□□□□□□□□□□□□

□2□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□[1] □

```
typedef char* CHARS;
typedef CHARS const CPTR;   //□□char□□□□□□□
```

[1] □C++□□□□□□□□□□□□“□□□□”□□□□□□□□□□□□□□

```
typedef int Length;          //□□Length □int□□□
int i = 42;
Length l = 88;
i = l;                              //□□
```

```
I = i;                              //□□
```

□□□□CHARS□□□□□□□□□□□□□□□□2□□□□□□□□□□□

```
typedef char* const CPTR; //□□□□□char□□□□□□□
```

□□□□□□□□ const □□

```
typedef char* CHARS;
typedef const CHARS CPTR;   //□□char□□□□□□□
```

□□□□□□□□CHARS□□□□2□□□□□□□□□□□□□

```
typedef const char* CPTR;    //□□□□char□□□□□
```

□□□□□□□□□□□□□□□□volatile□□□□

□□□□□□□□□□□ & □□□□□□□□□□□□□□□□

```
void foo (int const& x);
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□

```
char* a, b;
```

□□□□□□□□□□C□□□□□□a□□□□□□□b□□□char□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□C++□□□□□□□□□□□□□<iostream>□□□□<stdio.h>□□□□□□□□□□<stddef.h>□□□□□□□□□□□□□□□□□□□□□<cstddef>□□□□□□□□□size_t□ptrdiff_t□□std:: □□□□□□□□□□□<stddef.h>□□□□□□□□□□□□□□□□std::size_t□□size_t□□□□□□□□□□

1.5 □□□□□

C++[illegible]1998[illegible]2002[illegible]
[illegible]C++[illegible]

[illegible]C++[illegible]C++[illegible]C++[illegible]13[illegible]

1.6 [illegible]

[illegible]
http://www.josuttis.com/tmplbook[illegible]

[illegible] David Vandevoorde [illegible]
http://www.vandevoorde.com/templates[illegible]

1.7 [illegible]

[illegible]——[illegible]

[illegible]E-mail[illegible]tmplbook@josuttis.com

[illegible]1[illegible] [illegible]

[illegible]C++[illegible]typename[illegible]

[illegible]Nicolai M.Josuttis[illegible]Object-Oriented Programming in C++[illegible]John Wiley[illegible]ISBN 0-470-84399-3[illegible]C++[illegible]C++[illegible]

[illegible]

[illegible]C++[illegible]quicksort[illegible]

[illegible]

1.[illegible]

2.[illegible]Object[illegible]void*[illegible]common base class[illegible]

3.[illegible]

[illegible]C[illegible]Java[illegible]

1.[illegible] [1] [illegible]

2.[illegible]

3.[illegible]C[illegible]C++[illegible]“[illegible]”[illegible]“[illegible]”[illegible]

[illegible]

[illegible]C++[illegible]

[illegible]2[illegible] [illegible]

[illegible]

2.1 [illegible]

[illegible]

2.1.1 [illegible]

[illegible]

```
//basics/max.hpp
template <typename T>
inline T const& max (T const& a, T const& b)
```

```
{
}
   // □□a < b□□□□□b□□□□□a
   return a < b ? b : a;
```

□□□□□□□□□□□□“□□□□□□□□□”□□□□□□□□□□□□□□□□□a□b□□□□□□□□□□□□□□□□□□□□□□□□□T□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
template < comma-separated-list-of-parameters >
//template < □□□□□□□□□□ >
```

□□□□□□□□□□□□□□typename T□□typename□□□□□□□□□□□T□□□□□□□□□C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□4□□□

□□□□□□□□□□□□ T□□□□□□□□□□□□□□□□□□□□□□□□□ T□□□□□□□□□□□□□□□□□□T□□□T□□□□□operator<□□□a□b□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□class□□typename□□□□□□□□□□C++□□□□□□□□□□□□ typename □□□□□□□□□□□□□□□□□□□ class □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□max()□□□□□□□□□□□□

```
template <class T>
inline T const& max (T const& a, T const& b)
{
}
   // □□ a < b □□□□□ b □□□□□ a
   return a < b ? b : a;
```

□□□□□□□□□class□typename□□□□□□□□□□□□□□□□class□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□class□□□□□□□□□□□□□□□□class□□□□□□□□□□□□□□□T□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□typename□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□struct□□typename□

2.1.2 □□□□

□□□□□□□□□□□□max()□□□□□

```
//basics/max.cpp
#include <iostream>
#include <string>
#include ”max.hpp”
    int main()
    {
    int i = 42;
    std::cout <<“max(7,i) : “ << ::max(7,i) <<std::endl;
    double f1 = 3.4;
    double f2 = -6.7;
    std::cout <<“max(f1,f2): “ << ::max(f1,f2)
<<std::endl;
    std::string s1 =“mathematics”;
    std::string s2 =“math”;
    std::cout <<“max(s1,s2): “ << ::max(s1,s2)
<<std::endl;
}
```

□□□□□□□□max()□□□□3 □□□□□□□□□□□□□□□□□□□int□□□□□□double□□□□□□std::string□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
max(7,i):42
max(f1,f2):3.4
```

max(s1,s2):mathematics

[illegible]max()[illegible] :: [illegible]max()[illegible]std::max()[illegible][2] [illegible]

[illegible][3] [illegible] 3 [illegible]max()[illegible]max()[illegible]

```
int i = 42;
… max(7,i) …
```

[illegible]int[illegible]T[illegible]

```
inline int const& max (int const& a, int const& b)
{
}
   return a < b ? b : a;
   // [illegible] a < b [illegible] b [illegible] a
```

[illegible]instantiation[illegible]——[illegible]

[illegible]

[illegible]max()[illegible]double[illegible]std::string[illegible]max[illegible]

```
const double& max (double const&, double const&);
const std::string& max ( std::string const&,
                         std::string const&);
```

[illegible]

```
std::complex<float>  c1, c2;  //std::complex
operator <
...
max(c1,c2);  //
```

1.

2.

6

2.2

[4] （deductiion）

max() int T const& C++ T int T

```
template <typename T>
inline T const& max (T const& a, T const& b);
...
max(4,7)       //OK: int
max(4,4.2)     //ERROR: 1 T int 2 T double
```

3

1.

```
max ( static_cast<double>(4),4.2)     //OK
```

max<double>(4,4.2)　　　　　//OK

2.□□□□□□□□□□T□□□□

3.□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□

2.3 □□□□

□□□□□□□□□□□□

1.□□□□□□□□□□□□□□□□□□□□□□□□□□□□

template <typename T>　　　　　//T□□□□□

2.□□□□□□□□□□□□□□□□□□□□□□□□□□□

…max (T const& a, T const& b)　　　//a□b□□□□□

□□ □□□□□ [5] □□□□□□□□□□□“□□□□□□□□□□□□□”max()□□□

```
template <typename T1, typename T2>
inline T1 max (T1 const& a, T2 const& b)
{
  return a < b ? b: a;
}
…
max(4,4.2) //OK, □□1□□□□□□□□□□□□□□
```

□□□□□□□□□ max()□□C++□□□□□□“□□□□□□□□‘□□□□’”□□□□□□□□□□□□□tricky□□□□□□□□□□□□□15.2.4□□□□□□□□□□□□□□□□□42□66.66□□□□□□□□□66.66□□□□□□□66□□□□□□□□□□2□

[illegible] [6] [illegible]T1[illegible]T1 const&[illegible]

[illegible]

[illegible]

```
template <typename T>
inline T const& max (T const& a, T const& b);
…
max<double>(4,4.2)    //[illegible]double[illegible]T
```

[illegible]3[illegible]

```
template <typename T1, typename T2, typename RT>
inline RT max (T1 const& a, T2 const& b);
```

[illegible] [7] [illegible]RT[illegible]RT[illegible]

```
template <typename T1, typename T2, typename RT>
inline RT max (T1 const& a, T2 const& b);
…
max<int, double, double>(4,4.2)    //OK, [illegible]
```

[illegible]“[illegible]”[illegible]

```
template <typename RT, typename T1, typename T2>
inline RT max (T1 const& a, T2 const& b);
…
max<double>(4,4.2)    //OK: [illegible]double
```

[illegible]max<double>[illegible]RT[illegible]double[illegible]T1[illegible]T2[illegible]int[illegible]double[illegible]

[illegible] max()[illegible]max()[illegible]

[illegible]11[illegible]

2.4 [illegible]

[illegible]C++[illegible]B[illegible]

[illegible]

```
//basics/max2.cpp
//[illegible]int[illegible]
inline int const& max (int const& a, int const& b)
{
    return a < b ? b : a;
}
// [illegible]
template <typename T>
inline T const& max (T const& a, T const& b)
{
    return a < b ? b : a;
}
```

```
// [illegible]3[illegible]
template <typename T>
inline T const& max (T const& a, T const& b, T const& c)
{
   return ::max (::max(a,b), c);
}
int main()
{
::max(7, 42, 68);             // [illegible]3[illegible]
   ::max(7.0, 42.0);        // [illegible]max<double> ([illegible])
   ::max('a', 'b');          // [illegible]max<char> ([illegible])
   ::max(7, 42);              // [illegible]int[illegible]
   ::max<>(7, 42);            // [illegible] max<int> ([illegible])
   ::max<double>(7, 42); //[illegible]max<double> ([illegible])
   ::max('a', 42.7);        //[illegible]int[illegible]
}
```

[illegible]4[illegible]

```
max(7[illegible]42)                              //[illegible]int[illegible]
```

[illegible]max()[illegible]2[illegible]3[illegible]

```
max(7.0,42.0);                  //[illegible] max<double>([illegible])
max('a', 'b');                  //[illegible] max<char>([illegible])
```

[illegible]

```
max<>(7,42)                          //call max<int>([illegible])
```

[illegible]'a'[illegible]42.7[illegible]int[illegible]

```
max('a',42.7)                //[illegible]
```

[illegible]C[illegible]

```
//basics/max3.cpp
#include <iostream>
#include <cstring>
#include <string>
// [illegible]
template <typename T>
inline T const& max (T const& a, T const& b)
{
      return a < b ? b : a;
}
// [illegible]
template <typename T>
inline T* const& max (T* const& a, T* const& b)
{
      return *a < *b ? b : a;
}
// [illegible]C[illegible]
inline char const* const& max (char const* const& a,
                               char const* const& b)
{
      return std::strcmp(a,b) < 0 ? b : a;
}
int main ()
```

```
{
int a=7;
    int b=42;
    ::max(a,b); // max() □□□int□□□□□
    std::string s="hey";
    std::string t="you";
    ::max(s,t); // max() □□□std:string□□□□□□
    int* p1 = &b;
    int* p2 = &a;
    ::max(p1,p2); // max() □□□□□□□□□□□□□
    char const* s1 = "David";
    char const* s2 = "Nico";
    ::max(s1,s2); // max() □□□c□□□□□□□
}
```

□□□max()□□□□□C-string□□□□□□□□□□□□□□□□□□□□□C-strings□max()□□□□□□□□□□□□□□□□□□□□□□3□□□□max()□□□□□3□C-string□□□□□

```
//basics/max3a.cpp
#include <iostream>
#include <cstring>
#include <string>
// □□□□□□□□□□□ (□□□□□□□□□)
template <typename T>
inline T const& max (T const& a, T const& b)
{
```

```
        return a < b ? b : a;
    }
    // □□C□□□□□□ (□□□□□□□)
    inline char const* max (char const* a, char const* b)
    {
        return std::strcmp(a,b) < 0 ? b : a;
    }
    // □3□□□□□□□□□ (□□□□□□□□)
    template <typename T>
    inline T const& max (T const& a, T const& b, T const& c)
    {
        return max (max(a,b), c); //□□□□□max(a,b)□□□□□
                                  //□□□□□□□
    }
    int main ()
    {
        ::max(7, 42, 68); // OK
        const char* s1 = "frederic";
        const char* s2 = "anica";
        const char* s3 = "lucas";
        ::max(s1, s2, s3); // □□□
    }
```

□□□□□□□□3□C-strings□□max()□□□□□

return max (max(a,b),c);

□□□□□□□□□□□□□□C-strings□□□□□□max(a,b)□□□□□□□□□□□□□□□□□□□□□max□□□□□□□□□□□□□□□□□□□□□□□

[illegible] 3 [illegible] max()[illegible]int[illegible]max()[illegible]3[illegible]int[illegible]max()[illegible]2[illegible]int[illegible]max()[illegible]

```
//basics/max4.cpp
// [illegible]
template <typename T>
inline T const& max (T const& a, T const& b)
{
    return a < b ? b : a;
}
// [illegible]3[illegible]
template <typename T>
inline T const& max (T const& a, T const& b, T const& c)
{
    return max (max(a,b), c); //[illegible]int
                              //[illegible]
}
// [illegible]int[illegible]
inline int const& max (int const& a, int const& b)
{
    return a < b ? b : a;
}
```

[illegible] 9.2 [illegible]

2.5 [illegible]

- [illegible]
- [illegible]
- [illegible]
- [illegible]
- [illegible]
- [illegible]

[illegible]3[illegible] [illegible]

[illegible]Stack[illegible]

3.1 [illegible]Stack[illegible]

[illegible] Stack< >[illegible]6.3[illegible]

```
//basics/stack1.hpp
#include <vector>
#include <stdexcept>
template <typename T>
class Stack {
  private:
    std::vector<T> elems;    // [illegible]
  public:
```

```
    void push(T const&);      // □□□□
    void pop();                // □□□□
    T top() const;             // □□□□□□
    bool empty() const {      // □□□□□□□
        return elems.empty();
    }
};
template <typename T>
void Stack<T>::push (T const& elem)
{
    elems.push_back(elem);    // □elem□□□□□□□□
}
template<typename T>
void Stack<T>::pop ()
{
    if (elems.empty()) {
        throw   std::out_of_range("Stack<>::pop():   empty
  stack");
    }
    elems.pop_back();          //□□□□□□□□
}
template <typename T>
T Stack<T>::top () const
{
    if (elems.empty()) {
        throw   std::out_of_range("Stack<>::top():   empty
  stack");
```

```
        }
        return elems.back();       // [illegible]
    }
```

[illegible]Stack<>[illegible]C++[illegible]vector< >[illegible]
[illegible]

3.1.1 [illegible]

[illegible]
[illegible]T[illegible]

```
template <typename T>
class Stack {
       .
};
```

[illegible]class[illegible]typename[illegible]

```
template <class T>
class Stack {
  ...
};
```

[illegible]T [illegible]T[illegible]
[illegible]vector[illegible]push()[illegible]T[illegible]top()[illegible]
[illegible]T[illegible]

```
template <typename T>
class Stack {
    private:
        std::vector<T> elems;      //[illegible]
    public:
        Stack();                          //[illegible]
```

```
        void push(T const &);      //□□□□
        void pop();                //□□□□
        T top() const;             //□□□□□□
};
```

□□□□□□□Stack<T>□□□T□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Stack<T>□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ [8] □

```
template <typename T>
class Stack {
    .
    Stack (Stack<T> const&);                          //□□□□□□
    Stack<T>& operator= (Stack<T> const&);       //□□□□
□
    ...
};
```

□□□□□□□□□□□□□□□□□□□□□□Stack□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Stack□

3.1.2 □□□□□□□

□□ [9] □□□□□□Stack<T>□□□□□push()□□□□□□

```
template <typename T>
void Stack<T>::push (T const& elem)
{
    elems.push_back (elem);   //□□□□□elem□□□
                  //□□□□□
}
```

[illegible]vector[illegible]push_back()[illegible]vector[illegible]

[illegible]vector [illegible] pop_back()[illegible]“[illegible]pop()”[illegible]Tom Cargill [illegible][CargillExceptionSafety] [illegible] Sutter [illegible][SutterExceptional][illegible]Item 10[illegible] pop()[illegible] T [illegible]vector[illegible]

```
template<typename T>
T Stack<T>::pop()
{
    if (elems.empty() ) {
        throw std::out_of_range(“Stack<>::pop(): empty Stack”);
    }
    T elem = elems.back();          //[illegible]
    elems.pop_back();               //[illegible]
    return elem;                    //[illegible]
}
```

[illegible]vector[illegible]back()[illegible]pop_back()[illegible] stack [illegible]std::out_of_range[illegible]top()[illegible]stack[illegible]top()[illegible] [10] [illegible]

```
template<typename T>
T Stack<T>::top() const
{
    if (elems.empty()) {
```

```
            throw     std::out_of_range(“Stack::top();     empty
        Stack”);
        }
        return elems.back();        //[illegible]
}
```

[illegible]

```
template <typename T>
class Stack {
          ..
        void push (T const& elem) {
            elems.push_back(elem);    //[illegible]elem[illegible]
        }
          ..
};
```

3.2 [illegible]Stack[illegible]

[illegible]Stack<>[illegible]

```
//basics/stack1test.cpp
#include <iostream>
#include <string>
#include <cstdlib>
#include "stack1.hpp"
int main()
{
        try {
```

```
        Stack<int>           intStack;        // □□□□□int□□
    [11]
        Stack<std::string> stringStack;    // □□□□□□□□□□
        // □□int□
        intStack.push(7);
        std::cout << intStack.top() << std::endl;
        // □□string□
        stringStack.push("hello");
        std::cout << stringStack.top() << std::endl;
        stringStack.pop();
        stringStack.pop();
      }
      catch (std::exception const& ex) {
        std::cerr << "Exception: " << ex.what() <<
    std::endl;
        return EXIT_FAILURE;  // □□□□□□□ERROR□□
      }
  }
```

□□□□□□Stack<int>□□□□□□□□□□□□int□□□T□□□□intStack□□□□□□ Stack<int>□□□□□□□□□□□□ vector□□□□□ int□□□□□□□□□□□□□□□□□□□□□int□□□□□□□□□□□□□□□□Stack<std::string>□□□□□□Stack<std::string>□□□□□□□□□□□ vector□□□□□ std::string□□□□□□□□□□□□□□□□□□□□□std::string□□□□□□

□□□“□□□□□□□□□□□□□□□”□□□□□□□□□□□□□□□□□□□□□□□□□“□□□□□□□□□□”□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□operator<□□□□□□□□□□□□□□“□□
operator<□”□□□□□□□□□□□□operator<□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□push()□top()□□□□□□□int□□□□□string□□□□pop()□□□□□□□string□□

□□□□□□□□□□□□□□□□□□□□□□□□□□Stack<int>□□□□□□□□□□□□□□□

```
void foo(Stack<int> const& s)//□□s□int□□□Stack<int>
{
    Stack<int> istack[10];    //istack□□□10□int□□□□
    ...
}
```

□□□□□□□□□□□□□□□□□□□

```
typedef Stack<int> IntStack;
void foo(IntStack const& s)         //s□□□int□
{
    IntStack istack[10];      //istack□□□□10□int□□□□
    ...
}
```

C++□□□□□□□□□□□“□□□□”□□□□□□□□□□□□□□□□□□□□

```
typedef Stack<int> IntStack;
```

□□□IntSatck□Stack<int>□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□int□Stack□□□□□□□□□

```
Stack<float*>           floatPtrStack;    //□□□□□□□□□□□
Stack<Stack<int> >    intStackStack;    //□□□□□int□□□
```

□□□□□□□□□□□□□□□□□□□□□

[illegible]>[illegible]operator>>[illegible]

```
Stack<Stack<int>> intStackStack; //ERROR:[illegible]>>
```

3.3 [illegible]

[illegible][12][illegible]

[illegible] template<>[illegible]

```
template<>
class Stack<std::string> {
}
...
```

[illegible]T[illegible]

```
void Stack<std::string>::push (std::string const& elem)
{
    elems.push_back(elem);    //[illegible]elem[illegible]
}
```

[illegible]std::string[illegible]Stack<>[illegible]

```
//basics/stack2.hpp
#include <deque>
#include <string>
#include <stdexcept>
```

```
#include "stack1.hpp"
template<>
class Stack<std::string> {
  private:
    std::deque<std::string> elems;      // [illegible]
  public:
    void push(std::string const&);      // [illegible]
    void pop();                         // [illegible]
    std::string top() const;            // [illegible]
    bool empty() const {                // [illegible]
      return elems.empty();
    }
};
void Stack<std::string>::push (std::string const& elem)
{
    elems.push_back(elem);    // [illegible]elem[illegible]
}
void Stack<std::string>::pop ()
{
    if (elems.empty()) {
      throw std::out_of_range
              ("Stack<std::string>::pop(): empty stack");
    }
    elems.pop_back();           // [illegible]
}
std::string Stack<std::string>::top () const
{
```

```
        if (elems.empty()) {
            throw std::out_of_range
                      ("Stack<std::string>::top(): empty stack");
        }
        return elems.back();       // [illegible]
    }
```

[illegible]deque[illegible]vector[illegible]stack[illegible] [13] [illegible]prinmary template[illegible]

3.4 [illegible]

[illegible]

```
template <typename T1, typename T2>
class MyClass {
   ...
};
```

[illegible]

```
//[illegible]
template <typename T>
class MyClass<T,T> {
   ...
};
//[illegible]2[illegible]int
template<typename T>
class MyClass<T,int> {
```

```
   ...
};
//□□□□□□□□□□□□□□□□□□□
template<typename T1,typename T2>
class MyClass<T1*,T2*>{
   ...
};
```

□□□□□□□□□□□□□□□□□□□□

```
Myclass<int,float> mif;          //□□MyClass<T1,T2>
MyClass<float,float> mff;       //□□MyClass<T,T>
MyClass<float,int>    mfi;        //□□MyClass<T,int>
MyClass<int*,float*> mp;        //□□MyClass<T1*,T2*>
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
MyClass<int,int> m;          //□□:□□□□□□□MyClass<T,T>
                          //          □MyClass<T,int>
MyClass<int*,int*> m;       //□□:□□□□□□□MyClass<T,T>
                          //          □MyClass<T1*,T2*>
```

□□□□□2□□□□□□□□□□□□□□□□□□□□□□□□

```
template<typename T>
class MyClass<T*,T*> {
   ...
};
```

□□□□□□□□□□12.4□□

3.5 □□□□□□

[illegible] Stack<> [illegible]2[illegible] std::vector<>[illegible]

```
//basics/stack3.hpp
#include <vector>
#include <stdexcept>
template <typename T, typename CONT = std::vector<T> >
class Stack {
  private:
    CONT elems;                // [illegible]
  public:
    void push(T const&);       // [illegible]
    void pop();                // [illegible]
    T top() const;             // [illegible]
    bool empty() const {       // [illegible]
      return elems.empty();
    }
};
template <typename T, typename CONT>
void Stack<T,CONT>::push (T const& elem)
{
    elems.push_back(elem);     // [illegible]elem[illegible]
}
template <typename T, typename CONT>
void Stack<T,CONT>::pop ()
{
```

```
    if (elems.empty()) {
      throw   std::out_of_range("Stack<>::pop():   empty
  stack");
    }
    elems.pop_back();          // [illegible]
}
template <typename T, typename CONT>
T Stack<T,CONT>::top () const
{
    if (elems.empty()) {
      throw   std::out_of_range("Stack<>::top():   empty
  stack");
    }
    return elems.back();     // [illegible]
}
```

[illegible]

```
template <typename T,typename CONT>
void Stack<T,CONT>::push(T const& elem)
{
    elems.push_back(elem);   //[illegible]elem[illegible]
}
```

[illegible]stack[illegible]vector[illegible]stack[illegible]

```
template<typename      T,typename      CONT      =
std::vector<T> >
class Stack {
    private:
```

```
        CONT elems;        //□□□□□□
    ...
};
```

□□□□□□□□Stack□□□□□□□□□□□□□□□

```
//basics/stack3test.cpp
#include <iostream>
#include <deque>
#include <cstdlib>
#include "stack3.hpp"
int main()
{
    try {
      // int□:
      Stack<int> intStack;
      // double□□□□□std::deque□□□□□
      Stack<double,std::deque<double> > dblStack;
      // □□int□
      intStack.push(7);
      std::cout << intStack.top() << std::endl;
      intStack.pop();
      // □□double□
      dblStack.push(42.42);
      std::cout << dblStack.top() << std::endl;
      dblStack.pop();
      dblStack.pop();
  }
  catch (std::exception const& ex) {
```

```
        std::cerr << "Exception: " << ex.what() << std::endl;
        return EXIT_FAILURE; // [illegible]ERROR[illegible]
    }
}
```

[illegible]

Stack<double,std::deque<double> >

[illegible]“[illegible]double[illegible]std::deque<>[illegible]”[illegible]

3.6 [illegible]

- [illegible]
- [illegible]
- [illegible]
- [illegible]
- [illegible]
- [illegible]

[illegible]4[illegible] [illegible]

[illegible]value[illegible] stack [illegible]

4.1 [illegible]

[illegible]stack[illegible]stack[illegible]

[illegible]

[illegible]stack[illegible]

[illegible]

[illegible]

[illegible]

```
//basics/stack4.hpp
#include <stdexcept>
template <typename T, int MAXSIZE>
class Stack {
  private:
    T elems[MAXSIZE];          // [illegible]
    int numElems;              // [illegible]
  public:
    Stack();                   // [illegible]
    void push(T const&);       // [illegible]
    void pop();                // [illegible]
    T top() const;             // [illegible]
    bool empty() const {       // [illegible]
        return numElems == 0;
    }
    bool full() const {        // [illegible]
        return numElems == MAXSIZE;
    }
};
// [illegible]
template <typename T, int MAXSIZE>
```

```
Stack<T,MAXSIZE>::Stack ()
  : numElems(0)                    // □□□□□□□□
{
    // □□□□□□
}
template <typename T, int MAXSIZE>
void Stack<T,MAXSIZE>::push (T const& elem)
{
    if (numElems == MAXSIZE) {
        throw  std::out_of_range("Stack<>::push():  stack
    is full");
    }
    elems[numElems] = elem;    // □□□□
    ++numElems;                    // □□□□□□□
}
template<typename T, int MAXSIZE>
void Stack<T,MAXSIZE>::pop ()
{
    if (numElems <= 0) {
        throw  std::out_of_range("Stack<>::pop():  empty
    stack");
    }
    --numElems;                    // □□□□□□□
}
template <typename T, int MAXSIZE>
T Stack<T,MAXSIZE>::top () const
{
```

```
    if (numElems <= 0) {
        throw    std::out_of_range("Stack<>::top():    empty
  stack");
  }
  return elems[numElems-1];   // [illegible]
}
```

MAXSIZE[illegible]2[illegible]int[illegible]

```
  template<typename T, int MAXSIZE>
  class Stack {
        private:
              T elems[MAXSIZE];      //[illegible]
                 ...
  };
```

[illegible]push()[illegible]

```
template <typename T, int MAXSIZE>
void Stack<T, MAXSIZE>::push (T const& elem)
{
    if (numElems = = MAXSIZE ){
        throw std::out_of_range ("Stack<>::push():stack is
  full")
    }
    elems [numElems] = elem;   //[illegible]
    ++numElems;                //[illegible]
}
```

[illegible]

```
//basics/stack4test.cpp
#include <iostream>
```

```
#include <string>
#include <cstdlib>
#include "stack4.hpp"
int main()
{
    try {
        Stack<int,20>   int20Stack;   // □□□□20□int□□□□
        Stack<int,40>   int40Stack;   // □□□□40□int□□□□
        Stack<std::string,40> stringStack; // □□□40□string□
  □□□
        // □□□□□20□int□□□□
        int20Stack.push(7);
        std::cout << int20Stack.top() << std::endl;
        int20Stack.pop();
        // □□□□□40□string□□
        stringStack.push("hello");
        std::cout << stringStack.top() << std::endl;
        stringStack.pop();
        stringStack.pop();
  }
  catch (std::exception const& ex) {
        std::cerr   <<   "Exception:   "   <<   ex.what()   <<
  std::endl;
        return EXIT_FAILURE; // □□□□□□ERROR□□
  }
}
```

[illegible] int20Stack [illegible] int40Stack [illegible]

[illegible]

[illegible]

```
template<typename T = int, int MAXSIZE = 100>
class Stack {
    ...
};
```

[illegible]int[illegible]100[illegible]

[illegible]

4.2 [illegible]

[illegible]

```
//basics/addval.hpp
template<typename T, int VAL>
T addValue(T const& x)
{
    return x + VAL;
}
```

[illegible]STL[illegible]

```
std::transform (source.begin(), source.end(), //[illegible]
                                               //[illegible]
        dest.begin(),                          //[illegible]
        addValue<int,5>);                      //[illegible]
```

□□□□□□□□□□□□□□□□□□□□□□□□addValue()□□□int□□□□5□□□□source□□□□□□□□□□□□□□□□addValue()□□□□□□□□□□□□□□□dest□

□□□□□□□□□□□□□□□□addValue<int,5>□□□

```
std::transform (source.begin(), source.end(), //□□□□□□
                //□□□
            dest.begin(),               //□□□□□□□
         (int(*)(int const&)) addValue<int,5>);    //□□
```

□□□□□□□□□□C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□[CoreIssue115]□□□

4.3 □□□□□□□□□□□

□□

□□□□□□□□class-type□ [14] □□□□□□□□□□□□□□□

```
template<double VAT>       //ERROR:□□□□□□□□□□□□□□
double process (double v)
{
    return v * VAT;
}
template<std::string name>    //ERROR:□□□□□□□□□□□□□□
class MyClass {
  ...
};
```

[illegible]C++[illegible]

[illegible]

```
template<char const* name>
class MyClass {
};
MyClass<”hello”> x;          //ERROR:[illegible]”hello”
```

[illegible]

```
template <char const* name>
class MyClass {
};
char const* s = ”hello”;
MyClass<s> x;                 //s[illegible]
```

[illegible]

```
template <char const* name>
class MyClass {
};
extern char const s[] = ”hello”;
MyClass<s> x;            //OK
```

[illegible]s[illegible]“hello”[illegible]

[illegible]8.3.3[illegible]13.4[illegible]

4.4 [illegible]

- [illegible]

• [illegible]class [illegible]string[illegible]

[illegible]

[illegible]5[illegible] [illegible]

[illegible]typename [illegible] [15] [illegible]template template parameters[illegible] [16] [illegible]C++[illegible]

5.1 [illegible]typename

[illegible]C++[illegible]typename[illegible]

```
template <typename T>
class MyClass {
      typename T::SubType * ptr;
      ...
};
```

[illegible]2[illegible]typename[illegible]SubType[illegible]T[illegible]ptr[illegible]T::SubType[illegible]

[illegible]typename[illegible]SubType[illegible]

```
T::SubType * ptr
```

[illegible]T[illegible]SubType[illegible]ptr[illegible]

[illegible]typename[illegible]9.3.2[illegible]

[illegible]typename[illegible]STL[illegible]

```
//basics/printcoll.hpp
#include <iostream>
// [illegible]STL[illegible]
template <typename T>
void printcoll (T const& coll)
{
    typename T::const_iterator pos; // [illegible]coll[illegible]
    typename T::const_iterator end(coll.end()); // [illegible]
    for (pos=coll.begin(); pos!=end; ++pos) {
        std::cout << *pos << ' ';
    }
    std::cout << std::endl;
}
```

[illegible]T[illegible]STL[illegible]STL[illegible]const_iterator[illegible]

```
class stlcontainer {
    typedef ...iterator;            //[illegible]
    typedef ...   const_iterator;   //[illegible]
    ...
};
```

[illegible]T[illegible]const_iterator[illegible]typename[illegible]

```
typename T::const_iterator pos;
```

.template[illegible]

[illegible]typename[illegible]bitset[illegible]

[illegible]

```
template <int N>
void printBitset (std::bitset<N> const& bs)
{
    std::cout<<bs.template
to_string<char,char_traits<char>,
                        allocator<char> >();
}
```

[illegible].template[illegible]template[illegible]bs.template[illegible]<[illegible]<[illegible]bs[illegible]N[illegible]

[illegible].template[illegible]->template[illegible]9.3.3[illegible]

5.2 [illegible]this->

[illegible]x[illegible]this->x[illegible]x[illegible]

```
template <typename T>
class Base {
    public:
        void exit();
};
template <typename T>
```

```
class Derived : Base<T> {
    public:
        void foo() {
            exit();        //[illegible]exit()[illegible]
    }
};
```

[illegible]foo()[illegible]exit()[illegible]Base[illegible]exit()[illegible]exit()[illegible]

[illegible]9.4.2[illegible] this->[illegible]Base<T>::[illegible]this->[illegible]Base<T>::[illegible]

5.3 [illegible]

[illegible] Stack<>[illegible]

```
Stack<int> intStack1, intStack2; //int[illegible]
Stack<float> floatStack;         //float[illegible]
...
intStack1 = intStack2;           //OK:[illegible]
floatStack = intStack1;          //ERROR:[illegible]
```

[illegible]

[illegible]Stack<>[illegible]

```
//basics/stack5decl.hpp
template <typename T>
class Stack {
  private:
    std::deque<T> elems;         // [illegible]
  public:
    void push(T const&);         // [illegible]
    void pop();                  // [illegible]
    T top() const;               // [illegible]
    bool empty() const {         // [illegible]
      return elems.empty();
    }
    // [illegible]T2[illegible]
    template <typename T2>
    Stack<T>& operator= (Stack<T2> const&);
};
```

[illegible]

1.[illegible]T2[illegible]

2.[illegible]deque[illegible]

```
//basics/stack5 assign.hpp
template <typename T>
template <typename T2>
Stack<T>& Stack<T>::operator= (Stack<T2> const& op2)
{
    if ((void*)this == (void*)&op2) {    //[illegible]
```

```
        return *this;
    }
    Stack<T2> tmp(op2);              // □□□□□□□□□□
    elems.clear();                   //□□□□□□□
    while (!tmp.empty()) {           // □□□□□□□
        elems.push_front(tmp.top());
        tmp.pop();
    }
    return *this;
}
```

□□□□□□□□□□□□□□□□□□□□□□□T□□□□□□□□□□□□□□□□□□T2□□□□□□

```
template <typename T>
template <typename T2>
...
```

□□□□□□□□□□□□□□□□□□□op2□□top()□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□op2□□□□□□□□□□□top()□□□pop()□□□□□□□□□□□□□top()□□□deque□□□□□push_front□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□int□□□□□□float□□

```
Stack<int>       intStack;       //int□
Stack<float>     floatStack;     //float□
...
floatStack = intStack;           //OK:□□□□□□□□□□□□
                                 //  □int□□□□□float
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□floatStack□□□□□
float□□□□□□□□□□top()□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□[17]□□□□□□□□
□□□
□□□□□□□□□□□□

```
elems.push_front(tmp.top());
```

□□
tmp.top()□□□□□□□□□elems.push_front()□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□

```
Stack<std::string> stringStack;          //std::string□
Stack<float>       floatStack;           //float□
...
floatStack = stringStack;                //ERROR:std::string□□□
                                         //□□□float
```

□□
□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
//basics/stack6decl.hpp
template <typename T, typename CONT = std::deque<T> >
class Stack {
  private:
    CONT elems;                  // □□□□□□
  public:
    void push(T const&);         // □□□□
    void pop();                  // □□□□
```

```
        T top() const;                      // □□□□□□
        bool empty() const {          // □□□□□□□
           return elems.empty();
        }
        // □□□□□□T2□□□□□□□
        template <typename T2, typename CONT2>
        Stack<T,CONT>&     operator=     (Stack<T2,CONT2>
    const&);
    };
    □□□□□□□□□□□□□□□□
    //basics/stack6assign.hpp
    template <typename T, typename CONT>
    template <typename T2, typename CONT2>
    Stack<T,CONT>&
    Stack<T,CONT>::operator=   (Stack<T2,CONT2>   const&
op2)
    {
        if ((void*)this == (void*)&op2) {    // □□□□□□
           return *this;
        }
        Stack<T2,CONT2> tmp(op2);           // □□□□□□□□□□
        elems.clear();                            // □□□□□□□□□
        while (!tmp.empty()) {                // □□□□□□□
           elems.push_front(tmp.top());
           tmp.pop();
        }
        return *this;
```

}

□□□vector□□□□□□□

//□□vector□□□□□□int□

Stack<int,std::vector<int> > vStack;

...

vStack.push(42);

vStack.push(7);

std::cout << vStack.top() << std::endl;

□□□□□□□□□□□□□□□□□□□□□□□□push_front()□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□basics [18] □□□□□□“stack6”□□□□□

5.4 □□□□□□□

[19]

□□□□□□□□□□□□□□□□□□□□□□ stack □□□□□□□□□□□□□□□□□□□

□ stack □□

Stack<int,std::vector<int> > vStack; //□□vector□int□

□□
Stack□□□□

Stack<int,std::vector> vStack; //□□vector□int
□

□□□□□□□□□□□□□2□□□□□□□□□□□□□□□□stack□□□□□□
[20] □

```
//basics/stack7.decl.hpp
template <typename T,
          template <typename ELEM> class CONT =
      std::deque >
class Stack {
   private:
      CONT<T> elems;         // □□□□□□□
   public:
      void push(T const&);   // □□□□
      void pop();            // □□□□
      T top() const;         // □□□□□□
      bool empty() const {   // □□□□□□□
         return elems.empty();
      }
};
```

□□□□□□□□□2□□□□□□□□□□□□□□□□□

```
template <typename ELEM> class CONT
```

□□□□□std::deque<T>□□std::deque□□□□□□□□2□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
CONT<T> elems;
```

□□□□□□□□□□□□□□□□□□1□□□□□□□□2□□□

□□□□□□□□□□□□□□□□□□□□□□□typename□□□□□□class□□□□□□□CONT□□□□□□□□□□□□□□□□□□class□□□□□□□□□□□□□□

```
template <typename T,
            template <class ELEM> class CONT =
         std::deque>
```

```
                    //□□
class Stack {
    .
};
```

□□□□□□□□□□□□

```
template <typename T,
          template <typename ELEM> typename CONT
      = std::deque>
class Stack {                              //□□
    ...
};
```

□□□□□□□□□□□□“□□□□□□□”□□□□□□□□□□ELEM□□□□□□□□□□□□□□□□

```
template <typename T,
              template <typename> class CONT =
          std::deque >
class Stack {
    ...
};
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□2□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□push()□□□□□□

```
template <typename T, template <typename> class
CONT>
void Stack<T,CONT>::push (T const& elem)
{
    elems.push_back(elem);        //□elem□□□□□□□□
}
```

[illegible]

[illegible]

[illegible]Stack[illegible]std::deque[illegible]CONT[illegible]std::deque[illegible] A [illegible]CONT[illegible]B[illegible]A[illegible]B[illegible]B[illegible]

[illegible]std::deque[illegible]2[illegible]allocator[illegible]std::deque[illegible]CONT[illegible]

[illegible]CONT[illegible]

```
template <typename T,
                    template <typename ELEM,
                       typename               ALLOC              =
                 std::allocator<ELEM> >
                          class CONT = std::deque>
class Stack {
      private:
            CONT<T> elems;          //[illegible]
            ...
};
```

[illegible]ALLOC[illegible]

[illegible]Stack[illegible]

```
//basics/stack8.hpp
#ifndef STACK_HPP
#define STACK_HPP
```

```
#include <deque>
#include <stdexcept>
#include <memory>
template <typename T,
          template <typename ELEM,
                    typename = std::allocator<ELEM> >
                    class CONT = std::deque>
class Stack {
  private:
    CONT<T> elems;          // □□□□□□□
  public:
    void push(T const&);   // □□□□
    void pop();            // □□□□
    T top() const;         // □□□□□□
    bool empty() const {   // □□□□□□□
      return elems.empty();
  }
  // □□□□□□□T2□□□□□□□
  template<typename T2,
           template<typename ELEM2,
                    typename = std::allocator<ELEM2>
                  >class CONT2>
  Stack<T,CONT>& operator= (Stack<T2,CONT2>
const&);
};
template <typename T, template
<typename,typename> class CONT>
```

```
void Stack<T,CONT>::push (T const& elem)
{
    elems.push_back(elem);    // [illegible]
}
template<typename T, template
<typename,typename> class CONT>
void Stack<T,CONT>::pop ()
{
    if (elems.empty()) {
        throw std::out_of_range("Stack<>::pop(): empty
    stack");
    }
    elems.pop_back();         // [illegible]
}
template <typename T, template
<typename,typename> class CONT>
T Stack<T,CONT>::top () const
{
    if (elems.empty()) {
        throw std::out_of_range("Stack<>::top(): empty
    stack");
    }
    return elems.back();      // [illegible]
}
template <typename T, template
<typename,typename> class CONT>
```

```
    template <typename T2, template
<typename,typename> class CONT2>
    Stack<T,CONT>&
    Stack<T,CONT>::operator= (Stack<T2,CONT2> const&
op2)
    {
      if ((void*)this == (void*)&op2) {    // □□□□□□
          return *this;
      }
      Stack<T2,CONT2> tmp(op2);            // □□□□□□□□□□
      elems.clear();                       // □□□□□□□□□
      while (!tmp.empty()) {               // □□□□□□□
          elems.push_front(tmp.top());
          tmp.pop();
      }
      return *this;
    }
    #endif // STACK_HPP
```

□□□□□□□□□□□□□□□□□□

```
    //basics/stack8test.cpp
    #include <iostream>
    #include <string>
    #include <cstdlib>
    #include <vector>
    #include "stack8.hpp"
    int main()
    {
```

```
    try {
        Stack<int>     intStack;            // int□
        Stack<float> floatStack;        // float□
        // □□int□
        intStack.push(42);
        intStack.push(7);
        // □□float□
        floatStack.push(7.7);
        // □□□□□□□□□□□□□
        floatStack = intStack;
        // □□float□
        std::cout << floatStack.top() << std::endl;
        floatStack.pop();
        std::cout << floatStack.top() << std::endl;
        floatStack.pop();
        std::cout << floatStack.top() << std::endl;
        floatStack.pop();
    }
    catch (std::exception const& ex) {
        std::cerr   <<   "Exception:   "   <<   ex.what()   <<
std::endl;
    }
// □□vector□□□□□□□int□
Stack<int,std::vector> vStack;
...
vStack.push(42);
vStack.push(7);
```

```
    std::cout << vStack.top() << std::endl;
    vStack.pop();
}
```

[illegible]

```
7
42
Exception: Stack<>::top(): empty stack
7
```

[illegible]

[illegible]8.2.3[illegible]15.1.6[illegible]

5.5 [illegible]

[illegible] int[illegible]double [illegible]“[illegible]”[illegible]undefined[illegible]

```
void foo()
{
    int x;          //x[illegible]
    int* ptr;       //ptr[illegible]
}
```

[illegible]

```
template <typename T>
void foo()
{
    T x;//[illegible]T[illegible]x[illegible]
```

```
}
```

□□0□□□false□□□bool□□□□□□□□□□□int()□□□□□□□□□0□□□

```
template <typename T>
void foo()
{
    T x = T();    //□□T□□□□□□x□□□□false
}
```

□□

```
template <typename T>
class MyClass {
    private:
        T x;
    public:
        MyClass() : x() {//□□x□□□□□□□□□□□□□□□□□
        }
        ...
};
```

5.6 □□□□□□□□□□□□□□□□

□□□

```
//basics/max5.cpp
#include <string>
// □□□□□□□
```

```
template <typename T>
inline T const& max (T const& a, T const& b)
{
    return   a < b   ?   b : a;
}
int main()
{
    std::string s;
    ::max("apple","peach");    // OK: [illegible]
    ::max("apple","tomato");   // ERROR: [illegible]
    ::max("apple",s);              // ERROR: [illegible]
}
```

[illegible]‘apple’[illegible]‘peach’[illegible]char const[6][illegible]‘tomato’[illegible]char const[7][illegible] max()[illegible]max()[illegible]

```
//basics/max6.cpp
#include <string>
// [illegible]
template <typename T>
inline T max (T a, T b)
{
    return   a < b   ?   b : a;
}
int main()
{
    std::string s;
```

```
    ::max("apple","peach");    // OK: □□□□□
    ::max("apple","tomato");   // OK: □□□decay□□□□□□□
    ::max("apple",s);          // ERROR: □□□□□
}
```

□□array-to-pointer□□□□□□□□□□□□□□□□□decay□□□□□□□□□□□□□□□□□□□□

```
//basics/refnonref.cpp
#include <typeinfo>
#include <iostream>
template <typename T>
void ref (T const& x)
{
    std::cout << "x in ref(T const&): "
              << typeid(x).name() << '\n';
}
template <typename T>
void nonref (T x)
{
    std::cout << "x in nonref(T): "
              << typeid(x).name() << '\n';
}
int main()
{
    ref("hello");
    nonref("hello");
}
```

[illegible] main [illegible]
[illegible] typeid [illegible] typeid [illegible]
std::type_info [illegible] lvalue [illegible] std::type_info [illegible] typeid [illegible]
[illegible] std::type_info [illegible] name() [illegible]
[illegible] C++ [illegible] name() [illegible] C++ [illegible]
[illegible] name() [illegible] typeid [illegible]
[illegible]

x in ref(T const&):   char[6]

x in nonref(T):   const char *

[illegible] [21] [illegible]

• [illegible]

• [illegible] B.2.2 [illegible]

• [illegible] std::string [illegible]

• [illegible]

```
template <typename T, int N, int M>
T const* max(T const (&a)[N], T const (&b)[M])
{
    return a < b ? b : a;
}
```

• [illegible]

[illegible] max() [illegible] 2.4 [illegible]
[illegible] max() [illegible] a<b [illegible]
[illegible] a<b [illegible]
std::string [illegible] C [illegible]

[illegible]11.1[illegible]

5.7 [illegible]

- [illegible]typename[illegible]
- [illegible]2[illegible]
- [illegible]
- [illegible]
- [illegible]“[illegible]”[illegible]std::deque[illegible]allocator[illegible]
- [illegible]
- [illegible]array-to-pointer[illegible]decay[illegible]

[illegible]6[illegible]

[illegible]

[illegible] 10 [illegible] C++[illegible]C++[illegible]

6.1 [illegible]

[illegible]（inclusion model）[illegible]

6.1.1 [illegible]

[illegible]C[illegible]C++[illegible]

• [illegible]class [22] [illegible](other type)[illegible].hpp[illegible].H[illegible].h[illegible].hh[illegible]hxx[illegible]

• [illegible]dot-C[illegible]dot-C[illegible].cpp[illegible].C[illegible].c[illegible].cc[illegible].cxx[illegible]

[illegible]

[illegible]

```
//basics/myfirst.hpp
#ifndef MYFIRST_HPP
#define MYFIRST_HPP
//[illegible]
template <typename T>
void print_typeof (T const&)
#endif //MYFIRST_HPP
```

print_typeof()[illegible]dot-C[illegible]

```
//basics/myfirst.cpp
#include <iostream>
#include <typeinfo>
#include“myfirst.hpp”
```

```
//[illegible]/[illegible]
template <typename T>
void print_typeof (T const& x)
{
    std::cout << typeid(x).name() << std::endl;
}
```

[illegible]typeid[illegible]5.6[illegible]

[illegible]dot-C[illegible]

```
//basics/myfirstmain.cpp
#include“myfirst.hpp”
//[illegible]
int main()
{
    double ice = 3.0;
    print_typeof(ice); //[illegible]double[illegible]
}
```

[illegible]C++[illegible] print_typeof()[illegible]

[illegible] print_typeof()[illegible]print_typeof()[illegible]double[illegible] myfirst.cpp [illegible]

6.1.2 [illegible]

[illegible]

```
#include“myfirst.cpp”
```

[illegible]myfirst.hpp [illegible]dot-C [illegible]myfirst.cpp[illegible]3[illegible]myfirst.cpp[illegible]myfirst.hpp[illegible]

```
//basics/myfirst2.hpp
#ifndef MYFIRST_HPP
#define MYFIRST_HPP
#include <iostream>
#include <typeinfo>
//[illegible]
template <typename T>
void print_typeof(T const&);
//[illegible]/[illegible]
template <typename T>
void print_typeof(T const& x)
{
    std::cout << typeid(x).name() << std::endl;
}
#endif //MYFIRST_HPP
```

[illegible]

[illegible]myfirst.hpp[illegible]<iostream>[illegible]<typeinfo>[illegible]<iostream>[illegible]

[illegible]

[illegible]

[illegible]“[illegible]”[illegible] C++[illegible]10[illegible]C++[illegible] C++[illegible]

[illegible]

6.2 [illegible]

[illegible]C++[illegible]C++[illegible]（explicit instantiation directive）[illegible]

6.2.1 [illegible]

[illegible]

```
//basics/myfirstinst.cpp
#include“myfirst.cpp”
```

//[illegible]double[illegible]print_typeof()

template void print_typeof<double>(double const&);

[illegible]template[illegible]

//[illegible]int[illegible]MyClass<>[illegible]

template MyClass<int>::MyClass();

//[illegible]int[illegible]max()

template int const& max(int const&, int const&);

[illegible]

//[illegible]int[illegible]Stack<>

template class Stack<int>

//[illegible]string[illegible]Stack<>[illegible]

template Stack<std::string>::Stack();

template void Stack<std::string>::push(std::string const&);

template std::string Stack<std::string>::top()const;

//[illegible]

template Stack<int>::Stack();

[illegible] print_typeof<int>[illegible] print_typeof<double> [23] [illegible] [24] [illegible]

[illegible]

[illegible]10[illegible]

6.2.2 [illegible]

[illegible]#include[illegible]myfirst.cpp[illegible]myfirstdef.hpp[illegible]6.1[illegible]Stack<>[illegible]

stack.hpp:

```
#ifndef STACK_HPP
#define STACK_HPP

#include <vector>

template <typename T>
class Stack {
  private:
    std::vector<T> elems;
  public:
    Stack();
    void push (T const&);
    void pop();
    T top() const;
};

#endif
```

stackdef.hpp:

```
#ifndef STACKDEF_HPP
#define STACKDEF_HPP

#include "stack.hpp"

template <typename T>
void Stack<T>::push (T const& elem)
{
    elems.push_back(elem);
}
...

#endif
```

[illegible]6.1 [illegible]

[illegible]#include[illegible]stackdef.hpp[illegible]#include [illegible] stack.hpp[illegible]dot-C[illegible]6.2[illegible] [25]

stacktest1.cpp:

```
#include "stack.hpp"
#include <iostream>
#include <string>

int main()
{
    Stack<int> intStack;
    intStack.push(42);
    std::cout << intStack.top() << std::endl;
    intStack.pop();

    Stack<std::string> stringStack;
    stringStack.push("hello");
    std::cout << stringStack.top() << std::endl;
}
```

stack_inst.cpp:

```
#include "stackdef.hpp"
#include <string>

// instantiate class Stack<> for int
template Stack<int>;

// instantiate some member functions of Stack<> for strings
template Stack<std::string>::Stack();
template void Stack<std::string>::push(std::string const&);
template std::string Stack<std::string>::top()
```

□6.2 □□□□□□□□□□□□□□□□□□□□□□□

6.3 □□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□C++□□□□□□□□□□□□□□□□□□□□□□□□exporting template□□□□□□□□□□□□□□C++□□□□□□□□□separation model□□

6.3.1 □□□export

[illegible]export[illegible]export[illegible]export[illegible]

```
//basics/myfirst3.hpp
#ifndef MYFIRST_HPP
#define MYFIRST_HPP
//[illegible]
export
template <typename T>
void print_typeof(T const&);
#endif   //MYFIRST_HPP
```

[illegible] [26] [illegible]myfirst3_hpp[illegible]export[illegible]

[illegible]export[illegible]export[illegible]myfirst.cpp[illegible]myfirst.cpp[illegible]exported[illegible]#include[illegible]myfirst3.hpp[illegible]export[illegible]export[illegible]

[illegible]export[illegible]

```
export template <typename T>
class MyClass {
    public:
```

```
        void memfun1();         //□□□□exported□□□□
        void memfun2() {        //□□□□□□□□□
                ...
        }
        void memfun3();         //□□□□□□□□□
        ...
};
template <typename T>
inline void MyClass<T>::memfun3()
{
    ...
}
```

□□□export□□□□□inline□□□□□□□□□□□□□□□□□export□□□□□□template□□□□□□□□□□□□□□□□

```
template <typename T>
class Invalid {
    public:
        export  void  wrong(T);         // □ □ □ export □ □ □ □
    template□□
};
export template<typename T>           //□□□□□□□□export□
inline
inline void Invalid<T>::wrong(T)
{
}
export template<typename T>
```

```
inline T const& max(T const& a, T const& b)//[illegible]export[illegible]inline
{
    return a < b ? b : a;
}
```

6.3.2 [illegible]

[illegible]exported template[illegible]export [illegible]

[illegible]C++[illegible]4[illegible]export[illegible][27] [illegible]export[illegible]C++[illegible]export[illegible]export[illegible]export[illegible]

[illegible]export [illegible]“[illegible]”[illegible]make[illegible]nmake[illegible]

[illegible]10[illegible]

[illegible]export[illegible][28] [illegible]export[illegible][29] [illegible]exported[illegible] included [illegible]

[illegible]exported[illegible]
[illegible]

6.3.3 [illegible]

[illegible]
[illegible]

```
//basics/myfirst4.hpp
#ifndef MYFIRST_HPP
#define MYFIRST_HPP
//[illegible]USE_EXPORT,[illegible]export
#if defined(USE_EXPORT)
#define EXPORT export
#else
#define EXPORT
#endif
//[illegible]
EXPORT
template <typename T>
void print_typeof(T const&)[illegible]
//[illegible]USE_EXPORT,[illegible]
#if !defined(USE_EXPORT)
#include“myfirst.cpp”
#endif
#endif   //MYFIRST_HPP
```

[illegible] USE_EXPORT[illegible]
[illegible]#include“myfirst.hpp”[illegible] USE_EXPORT[illegible]

//[illegible]

```
#define USE_EXPORT
#include“myfirst.hpp”
```

[illegible] USE_EXPORT [illegible] myfirst.hpp [illegible]#include[illegible]myfirst.cpp[illegible]

```
//[illegible]
#include“myfirst.hpp”
...
```

[illegible]

[illegible]#include[illegible].cpp[illegible]6.2[illegible]3[illegible]USE_EXPORT[illegible]6.2.2[illegible]

6.4 [illegible]

[illegible]inline [illegible]“[illegible]”[illegible]

[illegible]dot-C[illegible]#include[illegible]

[illegible] inline [illegible]

[illegible] inline[illegible] [30] [illegible]

6.5 [illegible]

[illegible]C++[illegible]precompiled header[illegible]

[illegible]C++[illegible]

[illegible]#included[illegible]

[illegible]N[illegible]N[illegible] N+1 [illegible]N[illegible]N[illegible]N[illegible]20%[illegible]200%[illegible]

[illegible]#include[illegible]#include[illegible]included[illegible]

```
#include <iostream>
#include<vector>
#include<list>
...
```

[illegible]

```
#include<list>
#include<vector>
...
```

[illegible]

[illegible]#include [illegible]
[illegible]std.hpp[illegible] [31] [illegible]

```
#include <iostream>
#include <string>
#include <vector>
#include <deque>
#include <list>
...
```

[illegible]std.hpp[illegible]

```
#include“std.hpp”
...
```

[illegible] std.hpp [illegible] [32] [illegible]make[illegible]

[illegible]std.hpp[illegible]core.hpp[illegible]core.hpp[illegible]std.hpp[illegible]

```
#include“std.hpp”
#include“core_data_hpp”
#include“core_algos.hpp”
```

...

[illegible] #include“std.hpp”[illegible]‘core.hpp’[illegible] #include“core.hpp”[illegible]std.hpp[illegible]

6.6 [illegible]

[illegible]

[illegible]syntactic constraint[illegible]semantic constraint[illegible] operator<[illegible]

concept[illegible]C++[illegible]random access iterator[illegible]default constructible[illegible]concept[illegible]concepts[illegible]concept[illegible]concept[illegible]concept[illegible]concept[illegible]C++[illegible]concept random access iterator[illegible]concept bidirectional iterator[illegible]concept[illegible]

6.6.1 [illegible]

[illegible]“class X has no member ‘fun’ ”[illegible] fun[illegible]run[illegible] C++[illegible] list<string>[illegible] greater<string>[illegible] greater<int>[illegible]

```
std::list<std::string> coll;
...
//[illegible]‘A’[illegible]
std::list<std::string>::iterator pos;
pos = std::find_if(coll.begiin(), coll.end(), //[illegible]
          std::bind2nd(std::greater<int>(),”A”) );//[illegible]
```

[illegible]

[illegible]GNU C++[illegible]

```
/local/include/stl/_algo.h: In function 'struct _STL::_List_iterator<_STL::basic
_string<char,_STL::char_traits<char>,_STL::allocator<char> >,_STL::_Nonconst_tra
its<_STL::basic_string<char,_STL::char_traits<char>,_STL::allocator<char> > > >
_STL::find_if<_STL::_List_iterator<_STL::basic_string<char,_STL::char_traits<cha
r>,_STL::allocator<char> >,_STL::_Nonconst_traits<_STL::basic_string<char,_STL::
char_traits<char>,_STL::allocator<char> > > >, _STL::binder2nd<_STL::greater<int
> > >(_STL::_List_iterator<_STL::basic_string<char,_STL::char_traits<char>,_STL:
:allocator<char> >,_STL::_Nonconst_traits<_STL::basic_string<char,_STL::char_tra
its<char>,_STL::allocator<char> > > >, _STL::_List_iterator<_STL::basic_string<c
har,_STL::char_traits<char>,_STL::allocator<char> >,_STL::_Nonconst_traits<_STL:
:basic_string<char,_STL::char_traits<char>,_STL::allocator<char> > > >, _STL::bi
nder2nd<_STL::greater<int> >, _STL::input_iterator_tag)':
/local/include/stl/_algo.h:115:   instantiated from '_STL::find_if<_STL::_List_i
terator<_STL::basic_string<char,_STL::char_traits<char>,_STL::allocator<char> >,
_STL::_Nonconst_traits<_STL::basic_string<char,_STL::char_traits<char>,_STL::all
ocator<char> > > >, _STL::binder2nd<_STL::greater<int> > >(_STL::_List_iterator<
_STL::basic_string<char,_STL::char_traits<char>,_STL::allocator<char> >,_STL::_N
onconst_traits<_STL::basic_string<char,_STL::char_traits<char>,_STL::allocator<c
har> > > >, _STL::_List_iterator<_STL::basic_string<char,_STL::char_traits<char>
,_STL::allocator<char> >,_STL::_Nonconst_traits<_STL::basic_string<char,_STL::ch
ar_traits<char>,_STL::allocator<char> > > >, _STL::binder2nd<_STL::greater<int>
>)'
testprog.cpp:18:   instantiated from here
/local/include/stl/_algo.h:78: no match for call to '(_STL::binder2nd<_STL::grea
ter<int> >) (_STL::basic_string<char,_STL::char_traits<char>,_STL::allocator<cha
r> > &)'
/local/include/stl/_function.h:261: candidates are: bool _STL::binder2nd<_STL::g
reater<int> >::operator ()(const int &) const
```

[illegible]

[illegible]/local/include/stl/_algo.h [illegible]testprog.cpp[illegible]18[illegible]_algo.h[illegible]115[illegible]find_if[illegible]

[illegible] 'no match for call' [illegible] 'candidate are' [illegible] const int& [illegible] 18 [illegible] std::bind2nd(std::greater<int>(),"A") [illegible] <int> [illegible] <int> [illegible] <std::string> [illegible]

[illegible] MyTemplate<YourTemplate<int>> [illegible] MyTemplate<T> [illegible] T=YourTemplate<int> [illegible]

Leor Zolman [illegible] STLFilt [illegible] STL [illegible] http://www.bdsoft.com/tools/stlfilt.html [illegible]

6.6.2 [illegible]

[illegible]

```
template <typename T>
void clear (T const& p)
{
      *p = 0;          //[illegible] T [illegible]
}
template <typename T>
void core (T const& p)
{
      clear(p);
```

```
}
template <typename T>
void middle (typename T::Index p)
{
    core(p);
}
    template <typename T>
void shell (T const& env)
{
    typename T::Index i;
    middle<T>(i);
}
class Client {
    public:
        typedef int Index;
};
Client main_client;
    int main()
{
    shell(main_client);
}
```

[illegible] shell()[illegible]middle()[illegible]core()[illegible]shell()[illegible]int[illegible]core()[illegible]int[illegible]middle() [illegible] Client::Index [illegible] int [illegible]dereference[illegible]

[illegible][StroustrupDnE][illegible]Bjarne Stroustrup[illegible]13.11[illegible][33][illegible]

[illegible]shell()[illegible]T::Index [illegible]

```
template <typename T>
inline void ignore(T const&)
{
}
template <typename T>
void shell (T const& env)
{
    class ShallowChecks {
        void deref(T::Index ptr) {
            ignore(*ptr);
        }
    };
    typename T::Index I;
    middle(i);
}
```

[illegible]T[illegible]T::Index[illegible]ShallowChecks[illegible]shell()[illegible]ShallowChecks [illegible]ignore()[illegible]tricks [34][illegible]

[illegible]

[illegible] concept [illegible]

[illegible]Concept Check Library[illegible]Boost[illegible]

[illegible]

[illegible]

6.6.3 [illegible]

[illegible] 6.6.1 [illegible]

[illegible]std::string[illegible]

```
_STL::basic_string<char,_STL::char_traits<char>,
                    _STL::allocator<char> >
```

[illegible]C++[illegible]10 000[illegible]

[illegible]

6.6.4 [illegible]

[illegible] [35] [illegible]

[illegible]tracer[illegible]

[illegible]

[illegible]

```
//basics/tracer.hpp
#include <iostream>
class SortTracer {
```

```
  private:
    int value;                          //[illegible]
    int generation;                     //[illegible]
    static long n_created;              //[illegible]
    static long n_destroyed;            //[illegible]
    static long n_assigned;             //[illegible]
    static long n_compared;             //[illegible]
    static long n_max_live;             //[illegible]
//[illegible]
static void update_max_live() {
    if(n_created-n_destroyed > n_max_live) {
        n_max_live = n_created – n_destroyed;
    }
}
public:
static long creations() {
    return n_created;
}
static long destructions() {
    return n_destroyed;
}
static long assignments() {
    return n_assigned;
}
static long comparisons() {
    return n_compared;
}
```

```
static long max_live() {
    return n_max_live;
}
public:
//□□□□
SortTracer (int v = 0) : value(v), generation(1) {
    ++n_created;
    update_max_live();
    std::cerr <<“SortTracer #“ << n_created
              <<”,    created    generation“    <<
          generation
              <<“    (total:“    <<    n_create    –
          n_destroyed
              <<“)\n”;
}
//□□□□□□
SortTracer (SortTracer const& b)
    : value(b.value), generation(b.generation + 1) {
        ++n_created;
        update_max_live();
        std::cerr <<“SortTracer #”<< n_created
              <<”,   copied   as   generation“   <<
          generation
              <<“    (total:“    <<    n_created    –
          n_destroyed
              <<“)\n”;
}
```

```
    //□□□□
    ~SortTracer() {
        ++n_destroyed;
        update_max_live();
        std::cerr <<“SortTracer generation“ << generation
                    <<“ destroyed (total:“
                    << n_created – n_destroyed <<“)\n”;
    }
    // □□□□□
SortTracer& operator= (SortTracer const& b) {
    ++n_assigned;
    std::cerr    <<    "SortTracer    assignment    #"    <<
n_assigned
                    << " (generation " << generation
                    << " = " << b.generation
                    << ")\n";
    value = b.value;
    return *this;
}
// □□□□□
friend bool operator < (SortTracer const& a,
                                SortTracer const& b) {
    ++n_compared;
    std::cerr    <<    "SortTracer    comparison    #"    <<
n_compared
                    << " (generation " << a.generation
                    << " < " << b.generation
```

```
                << ")\n";
        return a.value < b.value;
    }
    int val() const {
        return value;
    }
    };
```

□□□□□ value □□□□□ tracer □□□□□□□□□□□□□□□□□□□□□□generation□□

□□□□□□□□□□□□□□dot-C□□□□

```
//basics/traler.cpp
#include "tracer.hpp"
long SortTracer::n_created = 0;
long SortTracer::n_destroyed = 0;
long SortTracer::n_max_live = 0;
long SortTracer::n_assigned = 0;
long SortTracer::n_compared = 0;
```

□□□□□□□□□tracer□□□C++□□□□std::sort□□□□□□□□□□□

```
//basics/tracertest.cpp
#include <iostream>
#include <algorithm>
#include "tracer.hpp"
int main()
{
    // □□□□□□□:
```

```
SortTracer input[] = { 7, 3, 5, 6, 4, 2, 0, 1, 9, 8 };
// □□□□□:
for (int i=0; i<10; ++i) {
   std::cerr << input[i].val() << ' ';
}
std::cerr << std::endl;
// □□□□□□:
long created_at_start = SortTracer::creations();
long max_live_at_start = SortTracer::max_live();
long assigned_at_start = SortTracer::assignments();
long compared_at_start = SortTracer::comparisons();
// □□□□:
std::cerr << "---[ Start std::sort() ]--------------------\n";
std::sort<>(&input[0], &input[9]+1);
std::cerr << "---[ End std::sort() ]----------------------\n";
// □□□□:
for (int i=0; i<10; ++i) {
   std::cerr << input[i].val() << ' ';
}
std::cerr << "\n\n";
// □□□□□□□:
std::cerr << "std::sort() of 10 SortTracer's"
            << " was performed by:\n "
            << SortTracer::creations() - created_at_start
            << " temporary tracers\n "
            << "up to "
            << SortTracer::max_live()
```

```
            << " tracers at the same time ("
            << max_live_at_start << " before)\n "
            <<           SortTracer::assignments()           -
        assigned_at_start
            << " assignments\n "
            <<           SortTracer::comparisons()           -
        compared_at_start
            << " comparisons\n\n";
}
```

[illegible]“[illegible]”[illegible]std::sort()[illegible]

std::sort() of 10 SortTracer’s was performed by:

15 temporary tracers

up to 12 tracers as the same time (10 before)

33 assignments

27 comparisons

[illegible]15[illegible]tracer[illegible]tracer[illegible]

[illegible]tracer[illegible]tracer[illegible]sort()[illegible] = =[illegible]>[illegible]

6.6.5 oracles

[illegible]

[illegible]tracer[illegible]oracles[illegible]run-time analysis oracles[illegible] tracers——[illegible]inference engine[illegible] MELAS[illegible][MusserWangDynaVeri] [36] [illegible]

[illegible]oracles[illegible]oracles [illegible]oracles[illegible]

6.6.6 archetypes

[illegible]tracers[illegible]“[illegible]tracer”[illegible]tracer[illegible]archetype [illegible] archetype[illegible]concept[illegible]archetype[illegible]

6.7 [illegible]

[illegible]dot-C[illegible]one-definition rule[illegible]ODR[illegible]A[illegible]

[illegible]C++[illegible]C++[illegible]10[illegible]

[StroustrupDnE][illegible] Stroustrup [illegible]5[illegible]1[illegible]2002[illegible]5[illegible]

[illegible]concept[illegible] [illegible] #include [illegible]

[illegible]C++[illegible] [illegible]Jeremy Siek[illegible]Concept Check Library[illegible][BCCL][illegible]——[illegible]Boost[illegible] [illegible][Boost][illegible]

6.8 [illegible]

- [illegible]“[illegible]”[illegible]
- [illegible]
- [illegible]
- C++[illegible] export[illegible]
- [illegible]
- [illegible]
- [illegible]#include[illegible]

[illegible]7[illegible] [illegible]

[illegible]C++[illegible] [illegible]C++[illegible]C++[illegible]

7.1 “□□□”□□“□□□”

□C++□□□□□□□union [37] □□□□□□□□class type□□□□□□□□□□□□□□□□□□□□“□□class□”□□□□□□□class□□struct [38] □□□□□□□class type□□□□□□□□□□□□□□□□□class type□□□□□□union□□□“□□class□”□□□□□□union□□

□□□□□□□□□□□□□□□□□□□□□□□□

•□□□□□(class template)□□□□□□□□□□□□□□□□□□□□□□□□□□□□

•□□□□□□□□(template class)□□□□□□□□□□□□

□1□□□□□□□□□□□

□2□□□□□□□□□

□3□□□□□template-id [39] □□□□□

□□□□2□□□□□3□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□(template class)□

□□□□□□□□□□□□□(function template)□□□□□□□(member function template)□□□□□□□□□□(template function)□□□□□□□(template member function)□

7.2 □□□□□□

□□specialization□□

□□□□C++□□ 3.3□□□□□□□□□□□□□template<>□□□□□□

```
template <typename T1,typename T2>            //□□□□□□
class MyClass {
```

```
    ...
};
template<>                                          //[illegible]
class MyClass<std::string,float> {
    ...
};
```

[illegible]explicit specialization[illegible]

[illegible]3.4[illegible]partial specialization[illegible]

```
template <typename T>
class Myclass<T, T>{
    ...
};
template <typename T>                               //[illegible]
class MyClass<bool,T> {
    ...
};
```

[illegible]general template[illegible]primary template[illegible]

7.3 [illegible]

[illegible]declaration[illegible]definition[illegible]C++[illegible]

[illegible] C++[illegible]construct[illegible] C++[illegible]scope [40] [illegible]partial

classification[illegible]

```
class C;            //[illegible]C[illegible]
void f(int p);    //[illegible]f[illegible]p[illegible]
extern int v;      //[illegible]v[illegible]
```

[illegible]goto[illegible]

[illegible]C++[illegible]definition[illegible]“[illegible]class type[illegible]”[illegible] extern[illegible]

```
class C { };                      //[illegible]C[illegible]
void f(int p) {               //[illegible]f()[illegible]
        std::cout << p << std::endl;
}
extern int v = 1;             //[illegible]v[illegible]
int w;              //[illegible]extern[illegible]
```

[illegible]

```
template <typename T>
void func(T);
```

[illegible]

```
template <typename T>
class S {};
```

[illegible]

7.4 [illegible]

“C++[illegible]”[illegible]ODR[illegible]one-definition rule[illegible]A[illegible]ODR[illegible]ODR[illegible]

- [illegible]
- [illegible]class type[illegible]struct[illegible]union[illegible]

[illegible]translation unit[illegible]#include[illegible]

[illegible]linkable entity[illegible]

7.5 [illegible]

[illegible]

```
template <typename T, int N>
class ArrayInClass {
    public:
        T array[N];
};
```

[illegible]

```
class DoubleArrayInClass {
    public:
        double array[10];
};
```

[illegible]double[illegible]10[illegible]T[illegible]N[illegible]C++[illegible]

```
ArrayInClass<double,10>
```

[illegible]ArrayInClass[illegible]

[illegible] template-id[illegible]
[illegible]“[illegible]”[illegible]

[illegible] DoubleArrayInClass [illegible]
template-id [illegible]

```
int main()
{
    ArrayInClass<double,10> ad;
    Ad.array[0] = 1.0;
}
```

[illegible]template parámete[illegible]template
argument[illegible]“[illegible][41] ”[illegible]
[illegible]

•[illegible]template[illegible]
T[illegible]N[illegible]

•[illegible]double[illegible]10[illegible]
[illegible]“[illegible]” [42] [illegible]

[illegible]template-id[illegible]
[illegible]

[illegible]
[illegible]
[illegible]ArrayInClass[illegible]
[illegible]N[illegible]

[illegible]
[illegible]

```
template <typename T>
```

```
class Dozen {
    public:
        ArrayInClass<T,12> contents;
};
```

[illegible]T[illegible]1[illegible]T[illegible]2[illegible]T[illegible]

---

[1]. [illegible]

[2]. [illegible]std[illegible]string[illegible]C++[illegible]max()[illegible]std max()[illegible]

[3]. “one-entity-fits-all[illegible]”[illegible]“[illegible]”[illegible]

[4]. [illegible]deduction[illegible]

[5]. [illegible]C++[illegible]C++[illegible]13.3[illegible]

[6]. [illegible]

[7]. [illegible]——[illegible]

[8]. [illegible]C++[illegible]9.2.3[illegible]

[9]. [illegible]Stack<T>[illegible]

[10]. [illegible]vector[illegible]stack[illegible]stack[illegible]vector[illegible]vector[illegible]stack[illegible]

[11]. [illegible]int[illegible]

[12]. [illegible]

[13]. [illegible]deque[illegible]vector[illegible]stack[illegible]deque[illegible]deque[illegible]string[illegible]deque[illegible] ([illegible]C++[illegible]std::stack<>[illegible])[illegible]

[14]. [illegible]class-type[illegible]7[illegible]

[15]. [illegible]nested class[illegible]“[illegible]”[illegible]“[illegible]”[illegible]

[16]. [illegible]template template parameters[illegible]“[illegible]”[illegible]“[illegible]”[illegible]“[illegible]”[illegible]

[17]. [illegible]

[18]. [illegible]

[19]. [illegible]VC6[illegible]VC7[illegible]

[20]. [illegible]std:deque[illegible]

[21]. [illegible] C++[illegible][Standard98][illegible]

std::make_pair(“key”,”value”) //ERROR.[illegible][Standard98]

[illegible]make_pair()[illegible]make_pair()[illegible][Standard02][illegible]

[22]. [illegible]class[illegible]type[illegible]type[illegible]

[23]. [illegible]

[24]. [illegible]print_typeof<int>[illegible]

[25]. [illegible]VC6[illegible]

[26]. [illegible]export[illegible]

[27]. [illegible]Edison Design Group, Inc.(EDG)[illegible][EDG][illegible]VC6[illegible]VC7[illegible]export[illegible]

[28]. [illegible]“[illegible]”[illegible]

[29]. [illegible]“[illegible]”[illegible]

[30]. [illegible]

[31]. [illegible]

[32]. C++[illegible] std.hpp [illegible]#include<std>[illegible]#include[illegible]

[33]. [illegible]“[illegible]”[illegible]

[34]. [illegible]trick,[illegible]

[35]. [illegible]generic code[illegible]generic[illegible]“[illegible]”[illegible]generic programming[illegible]“[illegible]”[illegible]generic[illegible]“[illegible]”[illegible]generic programming[illegible]“[illegible]”[illegible]

[36]. □□David Musser□□C++□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□

[37]. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

[38]. □C++□□class□struct□□□□□□□□□□□□□□□class□□□□□□□□
private□□struct□□□□□□□□public□□□□□□□□□□□□C++□□□□□□□□□
□class□□□□□□□□□“plain old data(POD)”□C□□□□□□□□□□□□□□
struct□

[39]. □□□template-id□7.5□□

[40]. □□□scope□□□□□“□”□□□□□□“□□□”□□□□□□□□□□□

[41]. □□□□□□□□□argument□□□□□□□□□□□□actual parameter□□□
□□□□□□parameter□□□□□□□□□formal parameter□□

[42]. □□□□□□□□□□□□□□□□□10□□□□□□□□□

[illegible]2[illegible] [illegible]

[illegible]1[illegible]C++[illegible]C++[illegible]2[illegible]

[illegible]

[illegible]C++[illegible]C++[illegible]

- [illegible]
- [illegible]
- C++[illegible]
- [illegible]
- [illegible]
- [illegible]

[illegible]8[illegible] [illegible]

[illegible]

8.1 [illegible]

C++[illegible]13.6[illegible]

template<...parameters here...>

[illegible]

export template<...parameter here...>

[illegible]export[illegible]6.3[illegible]10.3.3[illegible]

[illegible] [1] [illegible]

```
template <typename T>
class List {                    //[illegible]
public:
   template <typename T2>        //[illegible]
   List (List<T2> const&);       //([illegible])
   ...
};
template <typename T>
template <typename T2>
List<T>::List(List<T2> const& b) //[illegible]
{                                //[illegible]
   ...
}
template <typename T>
int length(List<T> const&);      //[illegible]
                    //[illegible]
```

```
   class Collection {
   template <typename T>              //[illegible]
   class Node {                       //[illegible]
         ...
   };
   template <typename T>              //[illegible]
                                      //[illegible]
   class Handle;                      //[illegible]
         template <typename T>        //[illegible]
   T* allco() {                       //[illegible]
         ...
   }
   ...
};
template <typename T>                 //[illegible]
                                      //[illegible]
class Collection::Handle {            //[illegible]
   ...
};
```

[illegible] [2] [illegible] template<…>[illegible]

[illegible]Union[illegible]

```
template <typename T>
union AllocChunk {
      T object;
      unsigned char bytes[sizeof(T)];
```

```
};
```

[illegible]

```
template <typename T>
void report_top (Stack<T> const&, int number = 10);
    template <typename T>
void fill (Array<T>*, T const& = T() );//[illegible] [3]
                  //T()[illegible]0
```

[illegible]fill()[illegible]2[illegible]T[illegible] [4] [illegible]

```
class Value {
    public:
        Value(int);               //[illegible]
}[illegible]
void init (Array<Value>* array)
{
    Value zero(0);
    fill(array, zero);            //[illegible] [illegible]T()
    fill(array);                  //[illegible]=T()[illegible]T [illegible]
                  //Value[illegible]
}
```

[illegible]3[illegible]3[illegible] [5] [illegible]

[illegible]1[illegible]

[illegible]2[illegible]

[illegible]3[illegible]

[illegible]first-class[illegible]

[illegible]

```
template <int I>
class CupBoard {
      void open();
      class Shelf;
      static double total_weight;
};
template <int I>
void CupBoard<I>::open()
{
      ...
}
template <int I>
class CupBoard<I>::Shelf {
      ...
};
template <int I>
double CupBoard<I>::total_weight = 0.0;
```

[illegible]

8.1.1 [illegible]

[illegible]C++[illegible]

[illegible] [6] [illegible]

```
template <typename T>
class Dynamic {
  public:
    virtual ~Dynamic (); //OK[illegible]Dynamic[illegible]
    template <typename T2>
    virtual void copy (T2 const&);
                      //[illegible]Dynamic<T>[illegible]
                      //[illegible]copy()[illegible]
};
```

8.1.2 [illegible]

[illegible]12[illegible]class[illegible]

```
int C;
class C;//[illegible]space[illegible]
int X;
template <typename T>
class X;      //[illegible]X[illegible]
struct S;
template <typename T>
class S;      //[illegible]struct S[illegible]
```

[illegible]C[illegible]

```
extern“C++”template <typename T>
void normal();          //[illegible]
extern“C”template <typename T>
```

```
void invalid();              //[illegible]C[illegible]
extern“Xroma”template <typename T>
void xroma_link();           //[illegible]
                             //Xroma[illegible]
```

[illegible] static [illegible]

```
template <typename T>
void external();             //[illegible]
                             //[illegible]
                             //[illegible]external()[illegible]
   template <typaname T>
static void internal();  //[illegible]
                             //[illegible]
```

[illegible]

8.1.3 [illegible]

[illegible]

```
template <typename T> class Box;                  //[illegible]
template <typename T> class Box<T>;               //[illegible]
template <typename T> void translate(T*);         //[illegible]
template <typename T> void translate<T>(T*);  //[illegible]
```

[illegible] 12 [illegible]13.7[illegible]

8.2 [illegible]

[illegible]3[illegible]

[illegible]1[illegible]

[illegible]2[illegible]

[illegible]3[illegible]

[illegible]

```
template <typename, int>        //[illegible]
class X;
```

[illegible]

```
template <typename T,              //[illegible]2[illegible]3[illegible]
                    T* Root,              //[illegible]1[illegible]T
                    template<T*> class Buf>
     class Structure;
```

8.2.1 [illegible]

[illegible]typename[illegible]class[illegible] [7] [illegible]>[illegible]

[illegible]typedef[illegible]T[illegible]class T[illegible]T[illegible]class[illegible]

```
template <typename Allocator>
class List {
     class Allocator* allocator;           //[illegible]
     friend class Allocator;                 //[illegible]
     ...
};
```

[illegible]

8.2.2 [illegible]

[illegible] [8] [illegible]

- [illegible]
- [illegible]
- [illegible]

[illegible]13.4[illegible]

[illegible]typename[illegible]

```
template <typename T,                                  //[illegible]
          typename T::Allocator* Allocator>            //[illegible]
class List;
```

[illegible] 1 [illegible] typename [illegible] T[illegible] 2 [illegible] typename[illegible]5.1[illegible]9.3.2[illegible]typename[illegible]

[illegible]decay[illegible]

```
template<int buf[5]> class Lexer;      //buf[illegible]int*[illegible]
template<int* buf> class Lexer;        //[illegible]
```

[illegible]static[illegible]mutable[illegible]const[illegible]volatile[illegible]

```
template<int const length> class Buffer;
                    //[illegible]const[illegible]
```

```
template<int length> class Buffer;                //[illegible]
```

[illegible]

8.2.3 [illegible]

[illegible]placeholder[illegible]struct[illegible]union[illegible]

```
template <template<typename X> class C>    //[illegible]
void f(C<int>* p);
template <template<typename X> struct C>   //[illegible]
void f(C<int>* p);
template <template<typename X> union C>    //[illegible]
void f(C<int>* p);
```

[illegible]

[illegible]A[illegible]

```
template <template<typename T,
                   typename A = MyAllocator> class
          Container>
class Adaptation {
      Container<int> storage;
                     //[illegible]Container<int,MyAllocator>
      ...
};
```

[illegible]

```
template <template<typename T, T*> class Buf>
class Lexer {
      static char storage[5];
```

```
    Buf<char,&Lexer<Buf>::storage[0]> buf;
};
template <template<typename T> class List>
class Node {
    static T* storage;
            //[illegible]
    ...
};
```

[illegible]([illegible]T)[illegible]Adaptation[illegible]

```
template <template <typename, typename = MyAllocator> class Container>
class Adaptation
{
    Container<int> storage;
            //[illegible]Container<int, MyAllocator>
    ...
};
```

8.2.4 [illegible]

[illegible] 13.3 [illegible]

```
template <typename T, typename Allocator = allocator<T> >
class List;
            //[illegible]allocator<T>[illegible]Allocator,
```

```
                //[illegible]T
```

[illegible]

```
template <typename T1, typename T2, typename T3,
                typename T4 = char, typename T5 =
          char>
class Quintuple;           //[illegible]
template <typename T1, typename T2, typename T3 =
char,
                typename T4, typename T5>
class Quintuple;           //[illegible]
                  //T4[illegible]T5[illegible]
     template <typename T1 = char, typename T2,
typename T3,
                typename T4, typename T5>
class Quintuple;           //[illegible]T1[illegible]
                  //[illegible]T2[illegible]
```

[illegible]

```
template <typename T = void>
class Value;
template <typename T = void>
class Value;          //[illegible]
```

8.3 [illegible]

[illegible]

• [illegible]
template-id[illegible]

• [illegible]injected[illegible]P1[illegible]P2……[illegible]X[illegible]X[illegible]template-id[illegible]X<P1,P2,……>[illegible]9.2.3[illegible]

• [illegible]

• [illegible]11[illegible]

8.3.1 [illegible]

[illegible]

```
//details/max.cpp
template <typename T>
inline T const& max(T const& a, T const& b)
{
    return a < b ? b : a;
}
int main()
{
    max<double>(1.0, -3.0);      //[illegible]
    max(1.0, -3.0);              //[illegible]double
    max<int>(1.0,3.0);           //[illegible]<int>[illegible]
                                 //[illegible]int[illegible]
}
```

[illegible]11[illegible]

```
//details/implicit.cpp
template <typename DstT, typename SrcT>
inline DstT implicit_cast (SrcT const& x) //Srct□□□□□
{                                                    //□DstT□□
□
    return x;
}
int main()
{
    double value = implicit_cast<double>(-1);
}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ template<typename SrcT, typename DstT> □□□□□□ implicit_cast□□□□□□□□□□□□□□

□□

```
template <typename Func, typename T>
void apply (Func func_ptr, T x)
{
    fun_ptr(x);
}
template <typename T> void single(T);
template <typename T> void multi(T);
template <typename T> void multi(T*);
int main()
{
    apply(&single<int>, 3);        //□□
```

        apply(&multi<int>, 7);   //□□□&multi<int>□□□

    }

□□□□□□□apply()□□□□□□□□□□□□□□□□□&single<int>□□□□□□□□□□□□□□□□□□□□Func□□□□□□□□□□□□□□□□2□□□□&multi<int>□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Func□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□C++□□□□□□□□□□□□□□□□

```
template<typename T> RT1 test(typename T::X const*);

template<typename T> RT2 test(...);
```

□□□test<int>□□□□□1□□□□□□□□□□□□□□□int□□□□□□□□□□□X□□□□□2□□□□□□□□□□□□□□□□□□&test<int>□□□□□□□□□□□□□□□□□2□□□□□□□□□□□□□□□int□□□□1□□□□□□□□□□□□□&test<int>□□□□□□□□□□□SFINAE□□□□□□□□&test<int>□□□□□□□□□□□□□□

□ □ □ “ □ □ □ □ □ □ □ □ □ substitution-failure-is-not-an-error □SFINAE□”□□□RT1□RT2□□□□□□

```
typedef char RT1;

typedef struct { char a[2];} RT2;
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□constant-expression□□□□□T□□□□□□□□X□

```
#define type_has_member_type_X(T) (sizeof(test<T>(0)) == 1)
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□sizeof□□□□□□□□□□□1□test□□□□□□□□□□□1□char□□□□□□1□□□□□test□□□□□□□□□□□□2□□□□□□□□□□□□□□□□□char□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□constant-expression□□□□□□□□□□□test<T>(0)□□□□□□□test□□□□□□□□□□□□□T□□□□□□X□□□□□□□□□1□□□□□□□□□T□□□□□□X□□□□□□

[illegible]B[illegible]0[illegible]1[illegible]15[illegible]

SFINAE [illegible]C++[illegible]

```
template<int I> void f(int (&)[24/(4-I)]);
template<int I> void f(int (&)[24/(4+I)]);
int main()
{
    &f<4>;          //[illegible]0[illegible]SFINAE[illegible]
}
```

[illegible]2[illegible]0[illegible]

```
template<int N> int g() { return N; }
template<int* P> int g() { return *P;}
int main()
{
    return g<1>();          //[illegible]1[illegible]int*[illegible]
}                           //[illegible]SFINAE[illegible]
```

15.2.2[illegible]19.3[illegible]SFINAE[illegible]

8.3.2 [illegible]

[illegible]

[illegible]1[illegible]

[illegible]2[illegible] class [illegible][1] [illegible]typedef[illegible]

1 □□□“□□□□”□□□unnamed□David□□□□□□□unnamed means with no name□□□□

```
        struct { int x; } s;
        enum { e = 3 } c;
```

s □ c □□□□□ unnamed types□

□□□□□□□□□□□□□□□□□□□

```
template <typename T> class List {
    ...
};
typedef struct {
    double x, y, z;
} Point;
typedef enum { red, green, blue } *ColorPtr;
int main()
{
    struct Association                  //□□□□
    {
        int* p;
        int* q;
    };
    List<Association*> error1;    //□□□□□□□□□□□□□□□
    List<ColorPtr> error2;           //□□□□□□□□□□□□□□□
                          //□□□□typedef□□□□
                          //*ColorPtr□□□ColorPtr
    List<Point> ok;                       //□□□□□□□typedef□□
                          //□□□□□□
}
```

[illegible]

```
template <typename T>
void clear (T p)
{
      *p = 0;          //[illegible]*[illegible]T
}
int main()
{
      int a;
      clear(a);      //[illegible]int[illegible]*
}
```

8.3.3 [illegible]

[illegible]

• [illegible]

• [illegible]char[illegible]int[illegible]

• [illegible]&[illegible]&[illegible]

• [illegible]&[illegible]

• [illegible]pointer-to-member constant[illegible]&C::m[illegible]C[illegible]class [illegible]m[illegible]“[illegible]”[illegible]

[illegible]“[illegible]”[illegible]const[illegible]volatile[illegible]

[illegible]

```
template <typename T, T nontype_param>
class C;
C<int,33>* c1;                    //[illegible]
int a;
C<int*,&a>* c2;                   //[illegible]
void f();
void f(int);
C<void(*)(int),f>* c3;         //[illegible]
                   //[illegible]f(int),f[illegible]&[illegible]
class X {
    Public:
    int n;
    static bool b;
};
C<bool&, X::b>* c4;                 //[illegible]
                   //[illegible]
    C<int X::*, &X::n>* c5;       //[illegible]
    template<typename T>
    void templ_func();
C<void(), &templ_func<double> >* c6;
                 //[illegible]
```

[illegible]

[illegible]“[illegible]”[illegible]

[illegible]

- [illegible]
- [illegible]

• [illegible]

[illegible]

```
template <char const* str>
class Message;
extern char const hello[] = ”Hello World!”;
Message<hello>* hello_msg;
```

[illegible]extern[illegible]const[illegible]

4.3[illegible]13.4[illegible]

[illegible]

```
template<typename T, T nontype_param>
class C;
class Base {
    Public:
    int i;
} base;
class Derived : public Base {
} derived_obj;
C<Base*, &derived_obj>* err1          //[illegible]
                           //[illegible]
C<int&, base.i>* err2;                //[illegible].[illegible]
                           //[illegible]
int a[10];
C<int*, &a[0]>* err3;                 //[illegible]
                           //[illegible]
```

8.3.4 [illegible]

“[illegible]”[illegible]“[illegible]”[illegible]“[illegible]”[illegible] [9] [illegible]“[illegible]”[illegible] [10] [illegible]

[illegible]

```
#include <list>
    //List[illegible]
    //   namespace std {
    //          template <typename T,
    //                                       typename  Allocator  =
allocator<T> >
    //          class list;
    //   }
template <typename T1,
              typename T2,
              template<typename> class Container>
                  //Container[illegible]
class Relation {
public:
  ...
private:
  Container<T1> dom1;
  Container<T2> dom2;
};
int main()
{
  Relation<int,double,std::list> rel;
      //[illegible]std::list[illegible]2[illegible]
```

```
    ...
}
```

□□□□□□□□□□□std::list□□□□□□□□□□□2□□□□□□□□□□□□□allocator□□□□□□□□□□□□□□std::list□Container□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□Relation□□□

```
#include <memory>
template <typename T1,
          typename T2,
          template<typename T,
             typename = std::allocator<T> > class
          Container>
          //Container□□□□□□□□□□□□□□
class Relation {
  public:
    ...
  private:
    Container<T1> dom1;
    container<T2> dom2;
};
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□13.5□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□class□□□□□□□□□□□□□□□□□□□□□□□□□□ class □□□□□□□□□□□□□□□□□□“struct□□”□“union□□”□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□class□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

8.3.5 [illegible]

[illegible]typedef[illegible]typedef[illegible]

```
template <typename T, int I>
class Mix;
typedef int Int;
Mix<int, 3*3>* p1;
Mix<Int, 4+5>* p2; //p2[illegible]p1[illegible]
```

[illegible]

[illegible]1[illegible]

[illegible]2[illegible]

8.4 [illegible]

[illegible]“[illegible]”[illegible]

[illegible]1[illegible]

[illegible]2[illegible]

[illegible]

```
template <typename T>
class Node;
```

```
template <typename T>
class Tree {
    friend class Node<T>;
    ...
};
```

[illegible]

[11] [illegible]

```
template <typename T>
class Tree {
    friend class Factory;     //[illegible]Factory[illegible]
    friend class Node<T>;     //[illegible]Node[illegible]
                              //[illegible]
};
```

9.2.2[illegible]

8.4.1 [illegible]

[illegible]

```
template <typename T1, typename T2>
void combine(T1,T2);
class Mixer {
    friend void combine<>(int&, int&);
                    //[illegible]T1 = int&, T2 = int&
    friend void combine<int, int>(int, int);
                    //[illegible]T1 = int, T2 = int
    friend void combine<char>(char, int);
```

```
                    //□□□T1 = char, T2 = int
        friend void combine<char>(char&, int);
                    //□□□□□□□□□□combine()□□
        friend void combine<>(long, long) { ...}
                    //□□□□□□□□□□□□□□□□□□
};
```

□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□1□□□□□□□□□ [12] □□□ [13] □□□

□2□□

□□□□□□□□□□□□□□

```
void multiply (void*);                 //□□□□
template <typename T>
void multiply(T);                      □□□□
class Comrades {
    friend void multiply(int) { }
                    //□□□□□□□□□ □□multiply(int)
                    //□□□□□□□□□□□□□□□□
    friend void ::multiply(void*)
                    //□□□□□□□□□□
                    //□□□□multiply<void*>□□
    friend void ::multiply(int);
                    //□□□□□□□□
```

```
        friend void ::multiply<double*>(double*)
                            //[illegible]
                            //[illegible]
        friend void ::error() { }
                            //[illegible]
};
```

[illegible]

```
template <typename T>
class Node {
        Node<T>* allocate();
        ...
};
template <typename T>
class List {
        friend Node<T>* Node<T>::allocate();
        ...
};
```

[illegible]

```
template <typename T>
class Creator {
        friend void appear() {          //[illegible]appear()
        ...                             //[illegible]Creator[illegible]
                            //[illegible]
        }
```

```
};
Creator<void> miracle;          //□□□□□□□appear()
Creator<double> oops;           //□□□::appear()□2□□□
□
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ appear □□□□□□□□□ODR□□□□□□□A□□

□□□

```
template <typename T>
class Creator {
    friend void feed(Creator<T>*) {   //□□T□□□□□□□
                        //□::feed()□□
        ...
    }
};
Creator<void> one;      //□□::feed□Creator<void>*□
Creator<double> two;            // □ □ □ □
feed(Creator<double>*)
```

□□□□□□□□□□Creator□□□□□□□□□□□□□feed()□□□

□□9.2.2□□□11.7□□

8.4.2 □□□□

□□□

```
class Manager {
```

```
    template <typename T>
        friend class Task;
    template <typename T>
        friend void Schedule<T>::dispatch(Task<T>*);
    template <typename T>
        friend int ticket() {
            return ++Manager::counter;
        }
    static int counter;
};
```

[illegible]

[illegible]

8.5 [illegible]

[illegible]20[illegible]80[illegible]C++[illegible]C++[illegible]

[illegible]C++[illegible]C++[illegible]

[illegible]“[illegible]”[illegible]higher-order genericity[illegible]C++[illegible]

□□
□□□□□□□□□□□□□□□□□

□9□ □□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□C++□□□□□□□□□□ C++□□x*y□□x□y□□□□□□□□□□□□
□□□□□□□x□□□□□□□□□□□□□y□□□□□□x□□□□□

□□□□□□C++□□C□□□□□□□□□□□□C++□□□□□□□□□□
□□
□1□□□□□□□□□2□□□□□□□□□□3□□□□□□□□□□□□□□□□□□
C++□□□□□□□□□□□□□□□□□□□□

9.1 □□□□

C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□9.1□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□1□□□□□□□□□□□□□□□□□□□□□□□□ .□□>□□□□□□□□
□□□□□□□□□□□□□□□□this->count□□□□□□□□count□□□□□
□□□□□count□□□□□□□□□□□□

□2□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□T□□□□□
□□std::vector<T>::iterator□□□□□□□□□□T□□□□□typedef□□
□□□□int□□□std::vector<T>::iterator□□□□□□□□

□9.1 □□□□□

分　　类	说明和要点
标识符（Identifier）	一个只由字母、下划线和数字组成的不间断字符序列。它不能以数字开始，而且某些标识符也为实现所保留：你不能在你的程序中引入它们（另外，作为一条原则，你应该避免以下划线开头和使用两个连续的下划线）。“字母”这个概念在这里具有更广的外延：它还包含通用字符名称（Univercal Charalter Name, UCN），UCN 采用非字符的编码格式来存储信息
运算符 id（Operator-function-id）	在关键字 operator 后面紧跟一个运算符符号。例如，operator new 和 operator []。许多运算符都具有其他表示方法，例如，用于取址的单目运算符 operator&可以等价地写为 operator bitand[1]
类型转换函数 id (Conversion-function-id)	用来表示用户定义的隐式类型转换运算符。例如 operator int&，也可以写成 operator int bitand
模板 id（Template-id）	是一个模板名称，在它后面紧跟位于一对尖括号内部的模板实参列表。例如，List<T, int, 0>（严格地说，C++标准只允许简单的标识符作为 template-id 的模板名称。然而，这种规定或许是一种失误，实际上 operator-function-id 也应该可以作为 template-id 的模板名称，例如：operator+<X<int>>）
非受限 id（Unqualified-id）	广义化的标识符（identifier），它还可以是前面的任何一种（包括 identifier、operator-function-id，conversion-function-id、template-id）或者析构函数的名称（诸如~Date 或~List<T, T, N>）
受限 id（Qualified-id）	用一个类名或者名字空间名称对一个 unqualified-id 进行限定，也可以只使用全局作用域解析运算符（如：：f）对它进行限定。显然，这种名称本身也可以是多次受限的。这类例子有 ::X，S::x，Array<T>::y，::N::A<T>::z
受限名称（Qualified name）	标准中并没有定义这个概念。当需要引用基于受限查找（qualified, lookup）的名称时，我们使用了这个概念。明确而言，它是一个 qualified-id 或者在前面显式使用成员访问运算符（即 . 或－>）的 unqualified-id。这样的例子有 S::x，this->f，p->A::m 等。然而，虽然在某些上下文中 class_mem 隐式地等价于 this->class_mem，但是单独一个 class_mem（即前面没有－>等）就不是一个 qualified name，也就是说受限名称的成员访问运算符必须是显式给出的
非受限名称（Unqualified name）	它是一个除 qualified name 之外的 unqualified-id。这并不是一个标准概念，我们只是用它来表示调用非受限查找（unqulified lookup）时引用的名称

分　　类	说明和要点
名称（Name）	一个受限或者非受限的名称
依赖型名称（Dependent name）	一个（以某种方式）依赖于模板参数的名称。显然，显式包含模板参数的受限名称或者非受限名称都是依赖型名称。对于一个用成员访问运算符（ . 或者->）限定的受限名称，如果访问运算符左边的表达式类型依赖于模板参数，该受限名称也是依赖型名称。另外，对于 this->b 中的 b，如果是在模板中出现的，那么 b 也是一个依赖型名称。最后，对于形如 ident(x, y, z)的调用，如果其中有某个参数（表达式）所属的类型是一个依赖于模板参数的类型，那么标识符 ident 也是一个依赖型名称
非依赖型名称（Nondepeadent name）	一个不属于依赖型名称的名称，根据上面的描述，我们大体可以知道它的范围

[illegible]9.1[illegible]C++[illegible]

[illegible]

9.2 [illegible]

C++[illegible]1[illegible]2[illegible]C++[illegible]

[illegible]

```
int x;
class B {
  public:
    int i;
};
class D : public B {
};
void f(D* pd)
{
```

```
        pd->i = 3;          //[illegible]B::i
        D::x = 2;           //[illegible]::x
    }
```

[illegible]

```
    extern int count;                       //(1)
    int lookup_example (int count)    //(2)
    {
        if(count < 0) {
            int count = 1;                //(3)
            lookup_example(count);    //[illegible]count[illegible]3[illegible]
        }
        return count + ::count;         //[illegible]1[illegible]count
                            //[illegible](2),[illegible]2[illegible]count[illegible]1[illegible]
    }
```

[illegible]——[illegible]——[illegible]argument-dependent lookup[illegible]ADL [14] [illegible]ADL[illegible]max()[illegible]

```
    template <typename T>
    inline T const& max (T const& a, T const& b)
    {
        return a < b ? b : a;
    }
```

[illegible]“[illegible]”[illegible]

```
    namespace BigMath {
        class BigNumber {
```

```
        ...
    };
    bool operator < (BigNumber const&, BigNumber const&);
    ...
}
using BigMath::BigNumber;
void g(BigNumber const& a, BigNumber const& b)
{
    ...
    BigNumber x = max(a, b);
    ...
}
```

□□□ max()□□□□□□ BigMath □□□□□□□□□□□□□□□□□“□□□ BigNumber □□□□ operator<”□□□□□□□□□□□□□□□□□□□□□ C++□□□□□□□□□□□ADL□□□□□□□□□□□□□□□□□□□□□□□□□

9.2.1 Argument-Dependent Lookup□ADL□

ADL□□ADL□□□□□□□□□□□□□□max□□□□□□□□□□□□□ADL□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ADL□□□□□□□□□□associated class [15] □□□□□□associated namespace□□□□□□□□□□□□associated class□associated namespace□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□namespace□class□□□□□□□□□□□□□□class X□□□□□□□□associated class□associated namespace□□□X□X□□□□□class□namespace.

□□□□□□□□□□associated class□associated namespace□□□□

□□□□□□□□□□□□□□□□□□□□□

• □□□□□□□□□□□□□

• □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□“□□□□□□”□□

□□□associated class□associated namespace□

• □□□□□□□associated namespace□□□□□□□□□□namespace□

□□□□□□associated class□□□□□□□□

• □□class□□□□□□□□□□associated class□□□□□□class□□□□□□

□□□□□□□□□□□□□□□□associated namespace □□□□□ associated

class □□□namespace□□□□□□□□□□□□□□□□ [16] □□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□class□namespace□

• □□□□□□□□□□□□□□□□□□□□□□□□associated class □

associated namespace□

□□□X□□□□□□□□□□□□□□□□□associated anmespace □

associated calss □□□□□□□□X□□□associated namespace □

associated class□

□□□ADL□□□□□associated class□associated namespace□□□

□□using-

directives □using□□□□□□□□□□□□□□□□

```
//details /adl.cpp
#include <iostream>
namespace X {
    template<typename T> void f(T);
}
namespace N {
    using namespace X;
    enum E { e1 };
```

```
        void f(E) {
          std::cout << "N::f(N::E) called\n";
        }
    }
    void f(int)
    {
        std::cout << "::f(int) called\n";
    }
    int main()
    {
        ::f(N::e1);   // [illegible]ADL
        f(N::e1);       // [illegible]f();   ADL [illegible] N::f(),
    }                           //   [illegible] [17]
```

[illegible]ADL[illegible]N[illegible]using-directive[illegible]main()[illegible]X::f()[illegible]

9.2.2 [illegible]

[illegible]A[illegible]A[illegible]

```
    template <typename T>
    class C {
        ...
        friend void f();
        friend void f(C<T> const&);
        ...
```

```
};
void g(C<int>* p)
{
    f();            //f()[illegible]
    f(*p);          //f(C<int> const&)[illegible]
}
```

[illegible] f() [illegible] C++ [illegible]

[illegible] 11.7 [illegible] C++ [illegible] ADL [illegible]

[illegible] f() [illegible] ADL [illegible] f(*p) [illegible] C<int> [illegible] *p [illegible] C<int> [illegible] C<int> [illegible] 2 [illegible] f [illegible] [18] [illegible]

9.2.3 [illegible]

[illegible]

```
//details/inject.cpp
#include <iostream>
int C;
class C {
  private:
```

```
        int i[2];
    public:
        static int f() {
            return sizeof(C);
        }
};
int f()
{
        return sizeof(C);
}
int main()
{
    std::cout << "C::f() = " << C::f() << ","
                      << " ::f() = " << ::f() << std::endl;
}
```

[illegible]C::f()[illegible]C[illegible]::f()[illegible]C[illegible]int[illegible]

[illegible]

```
template <template<typename> class TT> class X {
};
template <typename T> class C {
        C* a;                  //[illegible]C<T>* a
        C<void> b;             //[illegible]
        X<C> c;                //[illegible]C[illegible]
        X<::C> d;              //[illegible] <: [illegible] [ [illegible]
```

```
    X< ::C> e;          //[illegible] < [illegible] [illegible]
};
```

[illegible]C[illegible]<[illegible]

9.3 [illegible]

[illegible] [19] ——[illegible]int* p = 0;[illegible]int[illegible]/[illegible] *[illegible]p[illegible]/[illegible] [illegible]0[illegible]/[illegible] ;[illegible]

[illegible]0[illegible]*[illegible]p[illegible]“=”[illegible]“0”[illegible]int[illegible] *p = 0[illegible]p[illegible]

9.3.1 [illegible]

[illegible]C++[illegible]C++[illegible]

[illegible]C++[illegible]

x*

[illegible]x[illegible]x[illegible]

identifier, type, x

symbol, *

[illegible]x[illegible]

identifier, nontype, x

symbol, *

[illegible]

[illegible]

X<1>(0)

[illegible]X[illegible]0[illegible]X<1>[illegible]X[illegible]

(X < 1) > 0

[illegible]X[illegible]1[illegible]true[illegible]false[illegible]1[illegible]0[illegible]0[illegible]C++[illegible]C[illegible]C++[illegible] < [illegible] < [illegible] < [illegible]

[illegible]

```
template<bool B>
class Invert {
  public:
    static bool const result= !B;
};
void g()
{
    bool test = Invert<(1>0)>::result;          //[illegible]
```

}

[illegible]Invert<[illegible]1>0[illegible]>[illegible]1[illegible]>[illegible]Invert <1>[illegible]0>::result [20] [illegible]

[illegible]3.2[illegible]template-id[illegible]

```
List<List<int> > a;
                    //[illegible]
```

[illegible] > [illegible]>>[illegible]maximum munch[illegible]C++[illegible]

[illegible]C++[illegible] >> [illegible]>[illegible]C++[illegible]C++[illegible]C++[illegible]13.1[illegible]

[illegible]maximum munch[illegible]

```
class X {
    .
};
List<::X> many_X;          //[illegible]
```

[illegible] <: [illegible]digraph[illegible] [21] [illegible] [[illegible]List [:X> many_X[illegible] < [illegible] :: [illegible]

```
List< ::X> many_X;
                //[illegible]
```

9.3.2 [illegible]

□□□
□□□□□□□□□□□□□12□□□□□□□□□□□□□□□□□□□□□

```
template <typename T>
class Trap {
  public:
    enum { x };          //(1)□□x□□□□□
};
template<typename T>
class Victim {
  public:
    int y;
    void poof() {
        Trap<T>::x*y;             //□2□□□□□□□□□
    }
};
template<>
class Trap<void> {                  //□□□□□□□□□
  public:
    typedef int x;               //□3□□□x□□□□
};
void boom(Victim<void>& bomb)
{
    bomb.poof();
}
```

□□□□□□2□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□Trap<T>::x□□□□□□□□□□□□□□□□□Trap□□□□□□□□□□□□□□
□1□□Trap<T>::x□□□□□□□□□□□□□□2□□□□□□□□□□T□void□

[illegible]Trap X<T>::x[illegible]Victim[illegible]Trap<T>::x[illegible]int[illegible]

C++[illegible]typename[illegible]typename[illegible]C++[illegible]typename[illegible]class[illegible] typename[illegible] typename[illegible]

[illegible]1[illegible]

[illegible]2[illegible]

[illegible]3[illegible]

[illegible]4[illegible]

[illegible]3[illegible]typename[illegible] [22] [illegible]

```
template<typename_1 T>
struct S : typename_2 X<T>::Base {
    S()    :    typename_3    X<T>::Base(typename_4
X<T>::Base(0) ) { }
    typename_5 X<T> f() {
        typename_6 X<T>::C *p;//[illegible]p[illegible]
        X<T>::D* q;                    //[illegible]
    }
    typename_7 X<int>::C * s;
};
struct U {
```

```
        typename₈ X<int>::C * pc;
    };
```

[illegible]typename[illegible]1[illegible]typename$_1$ [illegible]2[illegible]3[illegible]typename[illegible]3[illegible]typename[illegible]4[illegible]typename$_4$ [illegible] 0 [illegible]X<T>::Base[illegible]5[illegible]typename[illegible]X<T>[illegible]6[illegible]typename[illegible] typename [illegible] typename[illegible]7[illegible]typename[illegible]3[illegible]4[illegible]8[illegible]typename[illegible]

9.3.3 [illegible]

[illegible]C++[illegible] < [illegible] < [illegible]template[illegible]

```
template <typename T>
class Shell {
  public:
    template<int N>
    class In {
      public:
        template<int M>
        class Deep {
          public:
```

```
                    virtual void f();
                };
            };
        };
        template<typename T, int N>
        class Weird {
          public:
            void          case1(typename          Shell<T>::template
        In<N>::template Deep<N>* p) {
                p->template Deep<N>::f();      //□□□□□□□
            }
            void          case2(typename          Shell<T>::template
        In<N>::template Deep<N>& p){
                p.template Deep<N>::f();      //□□□□□□□
            }
        };
```

□□□□□□□□□□□□□□□□□□□□□□□::□->□.□□□□□□□□□□ [23] □□□□□□□□ template□□□ template-id□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ typename□□□□□□□□□□□□□

p.template Deep<N>::f()

p□□□□□□□□□□□T□□□□C++□□□□□□□□Deep□□□□□□□□□□□□□□□□□□□□□□ Deep □□□□□□□□□□□□□□□ template □□□□□□□□□□□□□□□□□p.Deep<N>::f()□□□□□□□□p.Deep□< N□>f()□□ template□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ case1 □ case2□□□□□□□□□□□

[illegible]template[illegible][24][illegible]template[illegible]

9.3.4 using-declaration[illegible]

using-declaration[illegible]using-declaration [illegible]using-declaration[illegible]using-declaration[illegible]

```
class BX {
  public:
    void f(int);
    void f(char const*);
    void g();
};
class DX : private BX {
  public:
    using BX::f;
};
```

[illegible]using-declaration[illegible]Bx[illegible]f[illegible]DX[illegible]f[illegible]using-declaration[illegible]DX[illegible]DX[illegible]BX::f[illegible]DX[illegible]BX::f [illegible]using-declaration[illegible] BX::f [illegible]C++[illegible]public/private/protected[illegible]C++[illegible]

```
class DX : private BX {
  public:
    BX::f;          //[illegible]
                    //[illegible]using BX::f[illegible]
};
```

[illegible]using-declaration[illegible]

```
template <typename T>
class BXT {
  public:
typedef T Mystery;
    template<typename U>
struct Magic;
};
template <typename T>
class DXTT : private BXT<T> {
public:
using typename BXT<T>::Mystery;
Mystery* p; //[illegible]typename[illegible]
};
```

[illegible]using-declaration[illegible]typename[illegible]C++[illegible]

```
template <typename T>
class DXTM : private BXT<T> {
  public:
    using BXT<T>::template Magic;       //[illegible]
```

```
        Magic<T>* plink;                    //[illegible]Magic[illegible]
                                //[illegible]
    };
```

[illegible] Magic[illegible]

9.3.5 ADL[illegible]

[illegible]

```
namespace N {
    class X {
        ...
    };
    template<int I> void select(X*);
}
void g(N::X* xp)
{
    select<3>(xp);          //[illegible]ADL
}
```

[illegible]select<3>(xp)[illegible]ADL[illegible]select()[illegible]<3>[illegible]xp[illegible]<3>[illegible]select()[illegible](select<3)>(xp)[illegible]

9.4 [illegible]

[illegible]“[illegible]”[illegible]

9.4.1 [illegible]

[illegible]

```
template <typename X>
class Base {
  public:
    int basefield;
    typedef int T;
};
class D1 : public Base<Base<void> > {       //[illegible]
  public:
    void f() { basefield =3; }
};
template<typename T>
class D2 : public Base<double> { //[illegible]
  public:
    void f() { basefield =7; }    //[illegible]
    T strange;                            //T[illegible]Base<double>::T[illegible]
[illegible]
};
```

[illegible]D2[illegible]strange[illegible]Base<double>::T[illegible]T[illegible]int[illegible]C++[illegible]

```
void g (D2<int*>& d2, int* p)
{
```

```
    d2.strange = p;        //[illegible]
}
```

[illegible] [25] [illegible]D2[illegible]

9.4.2 [illegible]

[illegible]C++[illegible]C++[illegible]C++[illegible]

```
template<typename T>
class DD : public Base<T> {            //[illegible]
  public:
    void f() { basefield = 0; }        //(1)problem...
};
    template<>         //[illegible]
class Base<bool> {
  public:
    enum { basefield = 42 };           //[illegible]2[illegible]tricky!
};
void g(DD<bool>& d)
{
    d.f();                             //(3)oops?
}
```

□□1□□□□□□□□□□□□□□□□□basefield□□□□□□□□□□□□□□□□□□
Base□□□□□□□□□Base□□□□□basefield□□□int□□□□□□□□□□□□□□□□
□□□Base□□□□□□□□□□□□□□□basefield□□□□□□1□□basefield□□□□□
□□□□□□□□□□□□□□□int□□□□□□□□□□□□□□□□□□□□□3□□□□□DD::f□□
□□□□□□□□□□□□□1□□□□□□□□□□□□□□□□2□□□DD<bool>□□□□□□
basefield□□□□□□□□□□□□□□□□□□□3□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□C++□□□□□□□□□□□□□□□□□□□□□□[26]
□□□□□□□□□□□□□□□□□□□□□□□□C++□□□□□□□1□□□□□□□□□□□□□□
□□□□□□□□□□□□basefield□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□3□□□□□□□□□DD<bool>□□□□
Base<bool>□□□Base<bool>□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□basefield□□□□□□□□□□

```
//□□□□□□1□
template<typename T>
class DD1 : public Base<T> {
  public:
    void f() { this->basefield = 0; }//□□□□□□
};
```

□□□□□□□□□□□2□□□□□□□□□□□□□□

```
//□□□□□□2□
template<typename T>
class DD2 : public Base<T> {
  public:
    void f() { Base<T>::basefield = 0; }
};
```

□□
□□□2□□□□□

[illegible]1[illegible]

```
template<typename T>
class B {
  public:
    enum E { e1 = 6, e2 = 28, e3 = 496 };
    virtual void zero(E e = e1);
    virtual void one(E&);
};
    template<typename T>
class D : public B<T> {
  public:
    void f() {
        typename D<T>::E e;       //this->E[illegible]
        this->zero();             //D<T>::zero()[illegible]
        one(e);                   //one[illegible]
                      //[illegible]
    }
};
```

[illegible]one(e)[illegible]one[illegible] e[illegible] D<T>::E[illegible]“[illegible]”[illegible]one[illegible]this->[illegible]this->[illegible]

[illegible]

```
//[illegible]3[illegible]
template<typename T>
class DD3 : public Base<T>{
```

```
    public:
      using Base<T>::basefield;      //（1）[illegible]
      void f() { basefield = 0; }//(2)[illegible]
  };
```

[illegible]2[illegible]1[illegible]using-declaration[illegible]using-declaration[illegible]

9.5 [illegible]

[illegible]Taligent[illegible]20[illegible]90[illegible]Bill Gibbons[illegible]Taligent[illegible]C++[illegible]Taligent[illegible]C++[illegible]C++[illegible]

[illegible]Tom Pennello[illegible]Metaware[illegible]Stroustrup[illegible][StroustrupDnE][illegible]Pennello[illegible]1991 [illegible]C++[illegible] [27] [illegible]List{::X}[illegible]9.3.3[illegible]

“[illegible] 9.4.2 [illegible]”[illegible] C++[illegible]1993[illegible]Bjarne Stroustrup[illegible]1994[illegible][StroustrupDnE][illegible] 1997 [illegible] C++[illegible]

[illegible] [28] [illegible] STL [illegible] C++ [illegible]

[illegible] 9.4.1 [illegible]

Andrew Koenig [illegible] ADL [illegible] ADL [illegible] koenig [illegible] a+b [illegible] N::operator+(a, b) [illegible] using declaration [illegible] ADL [illegible] 10 [illegible] ADL [illegible] koenig [illegible]

[illegible]10[illegible]

[illegible] [29] [illegible] C++ [illegible]

[illegible] C++ [illegible] C++ [illegible]

10.1 On-Demand[illegible]

[illegible]C++[illegible] [30] [illegible] on-demand[illegible]C++[illegible]on-demand[illegible]

on-demand[illegible]

```
template<typename T> class C;//(1)[illegible]
C<int>* p = 0;                  //(2)[illegible]:[illegible]C<int>[illegible]
template<typename T>
class C {
  public:
    void f();                   //[illegible]3[illegible]
};                              //[illegible]4[illegible]
void g (C<int>& c)              //[illegible]5[illegible]
{
    c.f();                      //[illegible]6[illegible]
}                               //     [illegible]C::f()[illegible]
```

[illegible]1[illegible]2[illegible]g[illegible]C[illegible]6[illegible]f[illegible]

[illegible]C<void>[illegible]

```
C<void>* p = new C<void>;
```

[illegible]C<void>[illegible] X [illegible]T[illegible]C<X>[illegible]C<void>[illegible]C<void>[illegible]operator new [illegible]operator delete[illegible]

[illegible]C++[illegible]class[illegible]

```
template<typename T>
class C {
  public:
    C(int);          //[illegible]
                     //[illegible]
};
void candidate(C<double> const&);     //(1)
void candidate(int) { }               //(2)
int main()
{
    candidate(42);        //[illegible]
}
```

[illegible] candidate(42)[illegible]2[illegible]1[illegible]42[illegible]C<double>[illegible]2[illegible]1[illegible]C<double>[illegible]

10.2 [illegible]

[illegible]class[illegible]

[illegible]“[illegible]”[illegible]

[illegible] union[illegible]union [illegible]
[31] [illegible]“[illegible]”[illegible]

[illegible]

[illegible]

```
//details/lazy.cpp
template <typename T>
class Safe {
};
template <int N>
class Danger {
  public:
    typedef char Block[N]; // [illegible]N<=0[illegible]
};
template <typename T, int N>
class Tricky {
  public:
    virtual ~Tricky() {
```

```
        }
        void no_body_here(Safe<T> = 3);
        void inclass() {
            Danger<N> no_boom_yet;
        }
        // void error() { Danger<0> boom; }
        // void unsafe(T (*p)[N]);
        T operator->();
        // virtual Safe<T> suspect();
        struct Nested {
            Danger<N> pfew;
        };
        union { // [illegible]union
            int align;
            Safe<T> anonymous;
        };
    };
    int main()
    {
        Tricky<int, 0> ok;
    }
```

[illegible]main()[illegible]C++[illegible]“[illegible]”[illegible]assume the best[illegible]Danger[illegible]Block[illegible]typedef[illegible]N[illegible]0[illegible]N[illegible]0[illegible]no_body_here()[illegible]3[illegible]Safe[illegible] Safe<T>[illegible]

[illegible] error()[illegible]Danger<0>[illegible]Danger<0>[illegible]typedef[illegible]0[illegible]Block[0][illegible]error()[illegible]error()[illegible]unsafe(T (*p)[N])[illegible]N[illegible]

[illegible] main()[illegible]Tricky[illegible]int[illegible]T[illegible]0[illegible]N[illegible]Tricky[illegible] suspect()[illegible]inclass()[illegible]struct[illegible]Nested[illegible] Danger<0>[illegible] typedef[illegible]unsafe(T (*p)[N])[illegible]0[illegible] [32] [illegible]union[illegible]Danger<N>[illegible]Safe<T>[illegible] [33] [illegible]Danger<0>[illegible]

[illegible]operator->[illegible]operator->[illegible]class[illegible]main[illegible]Ticky<int, 0>[illegible]int[illegible]operator->[illegible] [34] [illegible]T[illegible]T*[illegible]operator->[illegible]operator->[illegible]operator->[illegible]int[illegible]operator->[illegible]

10.3 C++[illegible]

[illegible]

10.3.1 [illegible]

[illegible]9[illegible]POI[illegible]point of instantiation[illegible]1[illegible]two-phase lookup[illegible] [35] [illegible]1[illegible]2[illegible]

[illegible]1[illegible]ADL[illegible]

[illegible]2[illegible]POI[illegible] [36] [illegible]ADL[illegible]

10.3.2 POI

[illegible]C++[illegible]POI[illegible]POI[illegible]

```
class MyInt {
public:
   MyInt(int i);
};
MyInt operator – (MyInt const&);
bool operator > (MyInt const&, MyInt const&);
```

```
    typedef MyInt Int;
    template <typename T>
void f(T i)
{
    if (i > 0) {
        g(-i);
    }
}
//(1)
void g(Int)
{
    //(2)
    f<Int>(42);        //□□□
    //(3)
}
//(4)
```

□C++□□□□□□□f<Int>(42)□□□□□□□□MyInt□□T□□□□□□f□□□□□□POI□□2□□□□3□□□□□□□□□□□□□□□□□□□□□□POI□□□C++□□□□□□□::f<Int>(Int)□□□□□□□□□□□□□□1□□□□4□□□□□□□□□□□□□4□□□□□g(Int)□□□□□□□1□□□□□□□□□□4□□□□ g(-i)□□□□□□□□□□□□□□□□□1□□□□POI□□□□□g(-i)□□□□□□□□□g(Int)□□1□□□□□□□□□□□□□□□□□□□□□□□□□□□C++□□□POI□□□“□□□□□□□□□□□□□□□□□□□□□□□”□□□□□□□□□□□□□□□□4□□

□□□□□□□□□□□□□□□□□□□MyInt□□□□□□□□□□int□□□□□□□□□□□POI□□□□2□□□□□g(-i)□□□□□□ADL□□□□□□int□□□□□□□□□□□□□□□□□int□□□□□□□□ADL□□□□□□□□□□□g [37] □□□□□□□□□□□typedef□□□□□typedef□

```
typedef int Int;
```

□□□□□□□□□□□□□□ [38] □□□□□□□□□□POI□□□□□□□□□□□□□□□□□□□□□□

```
template<typename T>
class S {
  public:
    T m;
};
//(5)
unsigned long h()
{
    //(6)
    return (unsigned) long) sizeof(S<int>);
    //(7)
}
//(8)
```

□□□□□□□□□□□□6□□□7□□□□□POI□□□□□□□□□S<int>□□□ POI □□□□8□□□□□□□□□□□□ sizeof(S<int>)□□□□□□□□□□□□□□□□□8□□□□□□□□□□□S<int>□□□□□□□sizeof(S<int>)□□□8□□□□□□□□□□□□□□□□□□□□□□□□□□□□□POI□□□□□“□□5□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
template<typename T>
class S {
  public:
    typedef int I;
```

```
};
//(1)
template<typename T>
void f()
{
    S<char>::I var1 = 41;
    typename S<T>::I var2 = 42;
}
int main()
{
    f<double>();
}
//(2):(2a),(2b)
```

□□□□□□□□□□□□f<double>□POI□□□2□□□□□□□□□□□□□□□f()□□□□□□□□S<char>□□□□□□□□□□□□□□□□□POI□□□□1□□□□□□□□□□f()□□□□S<T>□□□S<T>□□□□□□□□□□□□□S<char>□□□□□□□POI□□□□□□□(2)□□□□□f<double>□□□□□□□□□□□□S<double>□□□□□□□□□□□□□□□□□□□□□□□□□□□□POI□□S<double>□POI□□□□□□□□□□□□□□□□□□□□□□□□□POI□□□□□POI□□f<double>□□□□□□□□□□□□□□□□POI□□□□□□POI□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□f<double>□POI□□□2b□□□□□□□□□□——□□2a□□——□□□□POI□□S<double>□POI□□□□□□□□□□□□S<double>□S<char>□POI□□□□□

□□□□□□□□□□□□□□□□□□□ POI□□□□□□□□□□□□□□□□□□□□□□□□□POI□□□□□□□□□POI□□□□□□□□□□□□□□□□POI□□□□□□□□□□□□□□□POI□□□□□□□□□□□□□□□□□□□□□ODR□□□□□□□□□□□□□□□POI□□□□□□□□□□□□□□□□□□□□□C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□C++□□□□□□□□□□POI□□□□□□□□□□□□□□□□□POI□□□□□□□□□
□□□□

□□□
□□□□□□□□□□□□□□POI□□□□□□□□□□□□□□C++□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□

10.3.3 □□□□□□□□

□□□ POI □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□POI□□□□□□□□□□□□□□□POI□□□
□□
□□□□#inlcude □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□

□□□□□POI□□□□□□□□□□□□□export□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□max()□□□
□□□□□

```
//□□□□1□
#inlcude <iostream>
export template<typename T>
T const& max(T const&, T const&)
    int main()
{
    std::cout << max(7,42) << std::endl;   //(1)
}
//□□□□2□
export template<typename T>
T const& max(T const& a, T const& b)
{
```

```
        return a < b ? b : a;       //(2)
    }
```

[illegible]1[illegible]1[illegible]int[illegible]T[illegible]POI[illegible]2[illegible]max[illegible]POI[illegible]

10.3.4 [illegible]

[illegible]1[illegible]

```
    //[illegible]1[illegible]
    #include<iostream>
    export template<typename T> T const& max(T const&,
T const&);
        namespace N {
        class I {
          public:
            I(int i) : v(i) { }
            int v;
        };
        bool operator < (I const& a, I const& b) {
            return a.v < b.v;
        }
    }
    int main()
    {
        std::cout <<max(N::I(7), N::I(42)).v<<std::endl;//(3)
    }
```

[illegible]3[illegible]POI[illegible]2[illegible]2[illegible]max[illegible] < [illegible] 1 [illegible]2[illegible]

[illegible] [39] [illegible]1[illegible]2[illegible]I[illegible]10.3.1[illegible]

[illegible]1[illegible]C++[illegible]1[illegible] ADL [illegible]——[illegible]2[illegible]

[illegible]2[illegible]POI[illegible]ADL[illegible]1[illegible]ADL[illegible]ADL[illegible]1[illegible]

[illegible]POI[illegible] [40] [illegible]

10.3.5 [illegible]

[illegible]

[illegible]1[illegible]

```
template <typename T>
void f1(T x)
{
    g1(x);          //(1)
}
    void g1(int)
{
}
    int main()
{
```

```
    f1(7);        //□□□□□□g1!
}                 //(2):f<int>(int)□POI
```

□□f1(7)□□□□f1<int>(int)□□□POI□□□□main()□□□□□□□□2□□□□□□□□□□□□□□□□□□□□g1□□□□□□□□□□□□f1□□□□□□□□□□□□□□□□□g1□□□□□□□□□□□□□□□□□□□□□□□□□f□□□□□□□□□x□□□□□□□□□□T□□□□□□□□□□(1)□□□□□□□□□□□□g1□□□□(1)□□□□□□g1□□□□□1□□□□□g1□□□2□□□□f1□POI□□□□□□□□□□□□□□□□□□g1□□□□g1□□□□□□□□int□□int□□□□□□□□□□□□□□□□□2□□□□□□g1□□□□□□□f1□POI□□□□2□□□□□□□□□□□□□□g1□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□g1□

□2□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□3□□□□□□□□□□□□□□

```
//□□ common.hpp
export template<typename T>
void f(T);
    class A {
};
class B {
};
class X {
  public:
    operator A() { return A(); }
    operator B() { return B(); }
};
//□□ a.cpp:
#include“common.hpp”
void g(A)
{
```

```
}
    int main()
{
    f<X>(X());
}
//□□ b.cpp:
#include“common.hpp”
void g(B)
{
}
    export template<typename T>
void f(T x)
{
    g(x);
}
```

□□□a.cpp□□ main()□□□□□f<X>(X())□□□□□□□ b.cpp□□□□□□□exported□□□□□□□□□□□□g(x)□□□X□□□□□□□□□□□□□□□□□□□□□□□□g()□□□□□□□□□□□□□□□□□□□□b.cpp□□□□□□□□□□□□□□□□□□□□□□□a.cpp□□□ADL□□□□□□□□□□□□□□□□1□□□□□g(B)□□□2□□□□□g(A)□□□□□□□□□□□□□□□□□□□□□□□□□g□□□□□□□□□□□□□□□

□□□□□□□ b.cpp □□□□ g(x)□□exported□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

10.4 □□□□□□

[illegible]C++[illegible]C++[illegible]——[illegible]C++[illegible]

[illegible]C++[illegible]

[illegible] ODR [illegible]

```
//[illegible]a.cpp
int main()
{
}
//[illegible]b.cpp
int main(){
}
```

C++[illegible]C++[illegible]

[illegible]

```
//[illegible]t.hpp
//[illegible]
template<typename T>
```

```
class S {
   public:
      void f();
};
      template<typename T>
void S::f()         //□□□□
{
}
void helper(S<int>*);
//□□□a.cpp□
#include”t.hpp”
void helper(S<int>* s)
{
      s->f();         //(1)S::f□□1□POI□□□□□□
}
//□□ b.cpp:
#include“t.hpp”
int main()
{
      S<int> s;
      helper(&s);;
      s.f();          //(2)S::f□□2□POI(□□□□)
}
```

□□□□□□□“□□□□□□□□□□□□□”□□□□□□□□□□□□□□ [41] □□□□□□□□□□□□□□□□POI□□□□□□□□□□□□□1□□□□2□□□

C++[illegible]C++[illegible]C++[illegible]3 [illegible]

[illegible]

10.4.1 [illegible]

[illegible]C++[illegible]Borland[illegible]C++[illegible]Microsoft [illegible]Windows[illegible]

[illegible]

[illegible]

- [illegible]N[illegible]
- [illegible]
- [illegible]

[illegible]

[illegible]spilled inlined functions [42] [illegible]virtual function dispatch tables

[43] [illegible]

[illegible]

10.4.2 [illegible]

[illegible]Sun Microsystems[illegible]C++[illegible]4.0[illegible]3[illegible]POI[illegible]3[illegible]

1.[illegible]

2.[illegible]——[illegible]

3.[illegible]

[illegible]

•[illegible]3[illegible]3[illegible]2[illegible]

•[illegible]

[illegible]

[illegible]C[illegible]

[illegible]
[illegible]POI[illegible]

[illegible]
[illegible]POI[illegible]

10.4.3 [illegible]

[illegible]C++[illegible]Cfront 3.0——[illegible]Bjarne Stroustrup[illegible]C++[illegible] [44] [illegible]Cfront[illegible]1[illegible]C[illegible]2[illegible]Cfront[illegible]C[illegible]Cfront[illegible]Cfront [illegible]Cfront[illegible]

1.[illegible]

2.[illegible]prelinker[illegible]

3.[illegible]

4.[illegible]3[illegible]

[illegible]3[illegible]“[illegible]”[illegible]

[illegible]Cfront[illegible]

• [illegible] Cfront [illegible]“[illegible]”[illegible]

• [illegible]

• [illegible] [45] [illegible]Cfron[illegible]t [illegible]Cfront[illegible]

[illegible]C++[illegible] [46] [illegible]Edison Design Group[illegible]EDG[illegible]HP[illegible]C++[illegible] [47] [illegible]EDG[illegible]C++[illegible] [48] [illegible]

EDG[illegible]instantiation request file[illegible]template information file[illegible] .ii [illegible] .ti[illegible]

1. [illegible]EDG[illegible] .ii[illegible].ti[illegible]

2. [illegible].ti[illegible] .ii[illegible]

3. [illegible] .ii[illegible]1[illegible]

4. [illegible] .ii[illegible]

[illegible].ii[illegible].ii[illegible]

[illegible]EDG[illegible]

10.5 [illegible]

[illegible]POI[illegible]explicit instantiation directive[illegible]template[illegible]

```
template<typename T>
void f(T) throw(T)
{
}
//4[illegible]
template void f<int>(int) throw(int);
template void f<>(float) throw(float);
template void f(long) throw(long);
template void f(char);
```

[illegible]11[illegible]

[illegible]

```
template<typenameT>
```

```
class S {
   public:
      void f() {
      }
};
template void S<int>::f();
template class S<void>;
```

[illegible]

[illegible]C++[illegible] #pragma[illegible]

[illegible]C++[illegible]

[illegible]

[illegible]

```
//[illegible] toast.hpp:
template<typenameT>
void toast(T const& x)
{
      ...
}
```

[illegible]

```
//[illegible]
#include“toast.hpp”
template void toast(float);
```

[illegible]toast<float>[illegible] [49] [illegible]

[illegible]2002[illegible]C++[illegible]

[illegible]C++[illegible]

```
//[illegible]1[illegible]
template<typename T> void f();    //[illegible]
                                  //[illegible]
void g()
{
    f<int>();
}
//[illegible]2[illegible]
template<typename T> void f()
{
}
template void f<int>();            //[illegible]
void g();
int main()
{
```

```
    g();
}
```

[illegible]

[illegible]“[illegible]”[illegible]

```
// [illegible]1:
template<typename T> void f()
{
}
template<> void f<int>();   // [illegible]
void g() {
    f<int>();
}
// [illegible]2:
template<typename T> void f()
{
}
template void f<int>();    // [illegible]
void g();
int main()
{
    g();
}
```

[illegible]C++[illegible]mangled

name [50] [illegible]

[illegible]extern[illegible]

```
// [illegible] 1:
template<typename T> void f()
{
}
extern template void f<int>(); // [illegible]
void g()
{
    f<int>();
}
```

10.6 [illegible]

[illegible]C++[illegible] C++[illegible]

[illegible]

[illegible]C++[illegible]

[illegible]Cfront[illegible] C++[illegible] C++[illegible] Cfront [illegible]

[illegible]“[illegible]”[illegible]“[illegible]”[illegible]Sun [illegible]HP, EDG[illegible]Cfront[illegible]

[illegible]Cfront[illegible]

```
// [illegible] template.hpp:
template<class T> // Cfront[illegible]typename
void f(T);
// [illegible] template.cpp:
template<class T> // Cfront[illegible]typename
void f(T)
{
}
//[illegible]app.hpp:
class App {
      ...
};
// [illegible] main.cpp:
#include "app.hpp"
#include "template.hpp"
int main()
{
      App a;
      f(a);
}
```

[illegible]Cfront [illegible]Cfront [illegible].h[illegible].c[illegible].cpp[illegible].C[illegible]

```
// [illegible] main.cpp:
#include "template.hpp"
#include "template.cpp"
#include "app.hpp"
static void _dummy_(App a1)
{
}
    f(a1);
```

[illegible]template.cpp[illegible]

[illegible]_dummy_[illegible]Cfront [illegible]

[illegible]C++[illegible]ADL[illegible] C++[illegible]export[illegible]

[illegible]11[illegible] [illegible]

[illegible]concat<std::string, int>(s,3) [illegible]C++[illegible]

[illegible]C++[illegible]

[illegible]

11.1 [illegible]

[illegible]“[illegible]”[illegible]“[illegible]T[illegible]”[illegible]-[illegible]

```
template<typename T>
T const& max(T const& a, T const& b)
{
    return a<b ? b : a;
}
    int g = max(1,1.0);
```

[illegible]int[illegible]max()[illegible]T[illegible]int[illegible]2[illegible]double[illegible]2[illegible]T[illegible]double[illegible]int[illegible]max[illegible] 2.4 [illegible]12[illegible]

[illegible]

```
template<typename T>
typename T::ElementT at (T const& a, int i)
{
    return a[i];
}
void f(int* p)
```

```
{
    int x = at(p,7);
}
```

□□□T□□□int*□□□□□□□□T□□□□□□□□□□□□□□□□□□□□□□□T::ElementT□□□int*□□□T□□□□□□□□□□C++□□□□□□□□□□□□□ [51] □□□□□□□□□□□□□□□□□at()□□1□□□□□□□□□□□□□□“□□at()□□□□□□□□”□□□□□□□□□□□□□□□□□□□□□□ C++□□□□□□□□□□□□□

```
void f(int* p)
{
    int x = at<int*>(p,7);
}
```

□□□□□□□□□-□□□□□□□□□□□□□□□□□□□□□□□□□□A□□□□□□□□□□□□□P□□□□□□□□□□□□□□□□□□□□□□T&□□□P□□□□□□□T□□A□□□□□□□□□□□□P□□□□□□□□□□A□□□□□□□□□□□□□□□□□□□□□□□□□□□□□decay [52] □□□□□□□□□□□□□□□□□□□const□volatile□□□□□□

```
template<typename T> void f(T);        //P□□T
template<typename T> void g(T&);        //P□□□T
double x[20];
int const seven = 7;
f(x);          //□□□□(□□f)□T□double*
g(x);          //□□□□(□□g)□T□double[20]
f(seven);      //□□□□□T□int.
g(seven);      //□□□□T□int const
f(7);          //□□□□□T□int
```

g(7);   //□□□□□T□int =>□□□□□□7□□□int&

□□□□ [53] f(x)□x□□□□□□□decay□double* □□□□□□□□T□□□□□□□f(seven)□□const□□□□□□□□□□□T□□□□int□□□□□□□g(x)□T□□□double[20]□□□□□□□decay□□□□□□g(seven)□□□□□□□int const□□□□□□□□□□□□□□□□□□□□□const□volatile□□□□□□□□□□□□T□□□□int const□□□□□□□□g(7)□□□□T□□□int□□□□□□□□□□□□□□□□□□□const□volatile□□□□□□□□□□□□□□□□□□□□□□□□7□□□□□int& □□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□decay□□

```
template<typename T>
T const& max(T const& a, T const& b);
```

□□□□□max(“Apple”,“Pear”)□□□□□□□□T□□□□char const* □□□□“Apple”□□□□char const[6]□□“Pear”□□□□char const[5]□□□□□□□□□□□□decay□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□T□□□□□□char[6]□char[5]□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□5.6□□

11.2 □□□□□□

□□□T□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
template<typename T>
void f1(T*);
template<typename E, int N>
void f2(E(&)[N]);
template<typename T1, typename T2, typename T3>
void f3(T1 (T2::*)(T3*) );
class S {
```

```
   public:
void f(double*);
};
void g(int*** ppp)
{
bool b[42];
f1(ppp);              //□□T□int**.
f2(b);                //□□E□bool□N□42.
f3(&S::f);            //□□T1=void,T2=S,T3=double.
}
```

□□□□□□□□□□□□□□□□□□□□declarators□□□□□□□□□□template-id □□

- □□□□□□□□□□□□□□□Q<T>::X□□□□□□□□□□□□□□□□T□
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□S<I+1>□□□□□□□□□□□□□I□□□□□□□□□□□□□□□int(&)[sizeof(S<T>)]□□□□□□□□□T□

□□□

```
//details/fppm.cpp
template <int N>
class X {
   public:
      typedef int I;
      void f(int) {
      }
```

```
};
template<int N>
void fppm(void (X<N>::*p)(typename X<N>::I) );
int main()
{
    fppm(&X<33>::f);      //□□□N□□□33
}
```

□□□□□fppm()□□□□□X<N>::I□□□□□□□□□□□□□□□□□□□□□□□□□X<N>::*p□□□□□□□□X<N>□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□N□□□□N□□□□□□□□□X<N>::I□□□□□□□□□□&X<33>::f□□□□□□□□□□□□□□-□□□□□□□□□□□

□□X□Y□

```
template<typename T>
void f(X<Y<T>,Y<T> >);
void g()
{
    f(X<Y<int>,Y<int> >() );      //□□
    f(X<Y<int>,Y<char> >() );     //□□□□□□□
}
```

□□□□□□□□□□□□□□T□□□□□f()□□2□□□X□□□□□□□□□□□□

11.3 □□□□□□□

□□□□□□□□□□□□□□□□□□-□□□□A□P□□□□□□□□□□□□□□□□□□□□□□□□□□1□□□□□□□□□□□□□□□□□□□□□□□□P□□□□□□□□□□□□□□□□□□□ f

□□□□□□ A □□□□□□□□□□□□□□□□□□□pf□□□□□□□□□□□□□

```
template<typename T>
void f(T, T);
void (*pf)(char,char) = &f;
```

□□□□□□□□P□□□void(T, T)□□A□void(char,char)□□char□□T□□□□□□□□□□□□□□□pf□□□□□“□□f<char>”□□□□

□□□□□□□□□□□□□□□□□□□□□□□

```
class S {
  public:
    template<typename T, int N> operator T[N]&();
};
```

□□□□□□□□□-□□□□A□P□□□

```
void f(int (&)[20]);
void g(S s)
{
    f(s);
}
```

□□□□□□□□S□□□int (&)[20]□□□□□□A□int[20]□□□□P□T[N]□□□□□□□int□□T□□20□□N□□□□□□□□□□□□

11.4 □□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□P□□□□□A□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

• □□□□□□□□□□□□□□□□□□□□□□□□□□□P□□□□□A□□□□□□const□□volatile□□□□

• [illegible] A [illegible]const[illegible]volatile[illegible]P[illegible]

• [illegible]P[illegible]A[illegible]A[illegible]P[illegible]A[illegible]

```
template<typename T>
class B {
};
template<typename T>
class D : public B<T> {
};
template<typename T> void f(B<T>*);
void g(D<long> dl)
{
    f(&dl);         //[illegible]long[illegible]T
}
```

[illegible]A[illegible]P[illegible]

11.5 [illegible]

[illegible]

```
template<typename T>
class S {
  public:
    S(T b) : a(b) {
    }
```

```
    private:
        T a;
};
S x(12);        //[illegible]12[illegible]T
```

11.6 [illegible]

[illegible]

```
template<typename T>
void init(T* loc, T const& val = T() )
{
        *loc = val;
}
```

[illegible]——[illegible]

```
class S {
public:
    S(int, int);
};
S s(0,0);
int main()
{
    init(&s, S(7,42) ); //[illegible]T[illegible]S[illegible]T()[illegible]
                                //[illegible]T()[illegible]
                                //[illegible]
}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□C++□□□□□□

```
template<typename T>
void f(T x = 42){
}
int main()
{
    f<int>();         //□□□T = int
    f();              //□□□□□□□□□□□42□□□T
}
```

11.7 Barton-Nackman□□

□1994□□John.J.Barton□Lee R.Nackman□□□□□□□□□□□□□□□□□□□□□restricted template expansion□□□□□□□□□□□□□□□1994□□□□□□□□□□□□□□□□□ [54] □□□□□□□□□

□□□□□□□□□□□□□□□□□□□□ Array□□□□□□□□□□□□□□□operator==□□1□□□□□□this□□□□2□□□□□□□□□□□□□□operator== □□□

```
template<typename T>
class Array {
  public:
    ...
};
template<typename T>
```

```
bool operator == (Array<T> const& a, Array<T> const& b)
{
    ...
}
```

[illegible]operator == [illegible]Barton [illegible]Nackman[illegible]

```
template<typename T>
class Array {
  public:
    ...
    friend bool operator == (Array<T> const& a,
                              Array<T> const& b) {
      return ArraysAreEqual(a,b);
    }
};
```

[illegible]float[illegible]Array[illegible] Barton [illegible] Nackman [illegible]operator ==(T,T)——[illegible]T[illegible]

[illegible]operator ==[illegible]Array<T> const&, Array<T> const&[illegible] ArraysAreEqual[illegible]

[illegible]Barton-Nackman[illegible]B[illegible]C++[illegible]Barton[illegible]Nackman[illegible]ADL[illegible]

```
class S {
};
template<typename T>
class Wrapper {
  private:
    T object;
  public:
    Wrapper(T obj) : object(obj) {          //[illegible]T[illegible]
                         //[illegible]Wrapper<T>
    }
    friend void f(Wrapper<T> const& a) {
    }
};
int main()
{
  S s;
  Wrapper<S> w(s);
  f(w);                        //[illegible]Wrapper<S>[illegible]w[illegible]
  f(s);                        //[illegible]Wrapper<S>[illegible]s[illegible]
```

}

□□□□□□□□□ f(w)□□□□□□□□□ f()□□□□ Wrapper<S>□□□□□□□□ □□□□□ Wrapper<S>□□□ w □□□□ [55] □□□□□□□ f(s)□□□□□□□□□ f(Wrapper<S> const&)□□□□□□□□□Wrapper<S>□□□s□□□S□□□ □□□□□□□□□□□□□□S□Wapper<S>□□□□□□□□□□□Wrapper<S>□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ f□□□□□□□□□ f□□□□□□ □□□□□

11.8 □□□□

□□□□□□□□□□□□□□C++□□□□□□□□C++□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□C++□□□□□

□□C++□□ □□□□□Bill Gibbons□□□□□□□□Taligent □□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□C++□□□□□□□□□□□□□□□□□ □Taligent□□□□□□□□□□□□□□□□□Barton-Nackman□□□□□□□□□□□□□□ □□□

□□□□□□□□□□□□□Barton-Nackman□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□Barton-Nackman □□□□□□□□16.5□□□□

□12□ □□□□□

□□□□□□□□□□□□□□□C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□ □□□□□□□

[illegible]C++[illegible]
[illegible]
C++[illegible]

12.1 [illegible]

[illegible]

```
template<typename T>
class Array {
   private:
      T* data;
      ...
   public:
      Array(Array<T> const&);
      Array<T>& operator = (Array<T> const&);
      void exchange_with (Array<T>* b) {
         T* tmp = data;
         data = b->data;
         b->data = tmp;
      }
      T& operator[] (size_t k) {
         return data[k];
      }
      ...
   };
   template<typename T> inline
   void exchange (T* a, T* b)
```

```
{
   T tmp(*a);
   *a = *b;
   *b = tmp;
}
```

□□□□□□□exchange()□□exchange_with□□□□□□□□□□□CPU□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ Array<T>□□□□□□□□□ Array<T>□□exchange_with□□exchange()□□□□□□□□□Array<T>□□□□□data□

12.1.1 □□□□

□ □ □ □ □ □ □ □ □ □ □ □ □ □ exchange_with() □ □ □ □ □ □ □ □ □ □ □ exchange()□□□□□□□□□□□□□□□□□□□□□□□□□□

1.Array□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

2.□□□□□□□□□□□□□□□□□□□

```
template <typename T>
void generic_algorithm(T* x, T* y)
{
   ...
   exchange(x,y);          //□□□□□□□□□□□
   ...
}
```

□□□□□□C++□□□quick_exchange()□□□□

```
template<typename T> inline
```

```
void quick_exchange(T* a, T* b)                    // (1)
{
   T tmp(*a);
   *a = *b;
   *b = tmp;
}
template<typename T> inline
void quick_exchange(Array<T>* a, Array<T>* b)   // (2)
{
   a->exchange_with(b);
}
void demo(Array<int>* p1, Array<int>* p2)
{
   int x, y;
   quick_exchange(&x, &y);                        // uses (1)
   quick_exchange(p1, p2);                        // uses (2)
}
```

quick_exchange()[illegible]int*[illegible]1[illegible]int[illegible]T[illegible]1[illegible]Array<int>[illegible]T[illegible]2[illegible]int[illegible]T[illegible]quick_exchange(p1, p2)[illegible]C++[illegible]2[illegible]1[illegible]2[illegible]

12.1.2 [illegible]

[illegible] quick_exchange() [illegible] Array<T>[illegible]

[illegible]Arrar<T>[illegible]

```
struct S {
      int x;
} s1, s2;
void distinguish (Array<int> a1, Array<int> a2)
{
   int* p = &a1[0];
   int* q = &s1.x;
   a1[0] = s1.x = 1;
   a2[0] = s2.x = 2;
   quick_exchange(&a1, &a2);   // [illegible]*p == 1
   quick_exchange(&s1, &s2);   // [illegible]*q == 2
}
```

[illegible]quick_exchange()[illegible]1[illegible]Array[illegible]p[illegible]2[illegible]Array[illegible]non-Array[illegible]struct[illegible]s1[illegible]s1[illegible]quick_[illegible]exchange()[illegible]Array<T>[illegible]

```
template<typename T>
void exchange(Array<T>* a, Array<T>* b)
{
      T* p = &(*a)[0];
```

```
        T* q = &(*b)[0];
        for (size_t k = a->size(); k-- != 0; ) {
            exchange(p++, q++);
        }
    }
```

[illegible] exchange()[illegible] Array<T>[illegible] exchange()[illegible] Array<Array<char> >[illegible] inline[illegible]

12.2 [illegible]

[illegible]

```
// details/funcoverload.hpp
template<typename T>
int f(T)
{
    return 1;
}
template<typename T>
int f(T*)
{
    return 2;
}
```

[illegible]int*[illegible]1[illegible]T[illegible]int[illegible]2[illegible]T[illegible] [illegible][56] [illegible]

[illegible]

```
//details/funcoverload cpp
#include <iostream>
#include "funcoverload.hpp"
int main()
{
    std::cout << f<int*>((int*)0) << std::endl;
    std::cout << f<int>((int*)0) << std::endl;
}
```

[illegible]1

2

[illegible]f<int*>((int*)0) [57] [illegible]f<int*>[illegible]int*[illegible]f[illegible]1[illegible]f[illegible]f<int*>(int*)[illegible]f<int*>(int**)[illegible]2[illegible](int*)0[illegible]int*[illegible]1[illegible]

[illegible]2[illegible]

12.2.1 [illegible]

[illegible] [58] [illegible]

1.[illegible]

2.[illegible]

3.[illegible]const[illegible]volatile[illegible]const volatile[illegible]

4.[illegible]

5.[illegible]

6.[illegible]

[illegible]

```
template<typename T1, typename T2>
void f1(T1, T2);
template<typename T1, typename T2>
void f1(T2, T1);
template<typename T>
long f2(T);
template<typename T>
char f2(T);
```

[illegible]

```
#include <iostream>
template<typename T1, typename T2>
void f1(T1, T2)
{
    std::cout << "f1(T1, T2)\n";
}
template<typename T1, typename T2>
void f1(T2, T1)
{
    std::cout << "f1(T2, T1)\n";
}
```

```
// □□□□□□□□□□□□
int main()
{
    f1<char, char>('a', 'b'); // □□□□□□
}
```

□□□□□□□□□□□□□□f1<T1 = char, T2 = char>(T1,T2)□□□□□□ f1<T1 = char, T2 =char>(T2, T1)□□□

```
// □□□□1:
#include <iostream>
template<typename T1, typename T2>
void f1(T1, T2){
    std::cout << "f1(T1, T2)\n";
}
void g()
{
    f1<char, char>('a', 'b');
}
// □□□□2:
#include <iostream>
template<typename T1, typename T2>
void f1(T2, T1)
{
    std::cout << "f1(T2, T1)\n";
}
extern void g();   // □□□□□□□ 1
```

```
int main()
{
      f1<char, char>('a', 'b');
      g();
}
```

[illegible]

```
f1(T2, T1)
f1(T1, T2)
```

12.2.2 [illegible]

[illegible]

```
#include <iostream>
template<typename T>
int f(T)
{
      return 1;
}
template<typename T>
int f(T*)
{
      return 2;
}
int main()
{
      std::cout << f<int*>((int*)0) << std::endl;
      std::cout << f<int>((int*)0) << std::endl;
}
```

□□□□□□□□□□□□□□□<int*>□<int>□□□□□□□□□□□□□□□□□□□
□□
□□□□□□□□□□□□□□□□□□□□□□□□main()□□□□□□□□□□

```
#include <iostream>
template<typename T>
int f(T)
{
    return 1;
}
template<typename T>
int f(T*)
{
    return 2;
}
int main()
{
    std::cout << f(0) << std::endl;
    std::cout << f((int*)0) << std::endl;
}
```

□□□□□□□□(f(0))□□□□□□□int□□□□int□□T□□□□□1□□□□□□□□□□
□□□2□□□□□□□□□□□□□□□□□□□□□□□□□1□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□

□2□□□((f ((int*) 0))□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□f<int*>(int*)□f<int>(int*)□□□□□□□□□□□□□□□□□□□□
□□□□□ int*□□□□□□□□□□□□□□□□□□□□□□□□□□□□□B□□□□□□□□□□
□□□□□□□□□□□□□□□□□□“□□□□□□□□□□□”□□□□□□□□□□□□□□□□□
2□□□□□□□□□□□□□□□□□□□□□□□□□□□□□1

2

12.2.3 [illegible]

[illegible] 2 [illegible] 1 [illegible] 1 [illegible] 2 [illegible]

[illegible] ft1 [illegible] ft2 [illegible] 1 [illegible] 1 [illegible] 2 [illegible] 2 [illegible] “[illegible]” [illegible]

1. [illegible] “[illegible]” [illegible]

2. [illegible] “[illegible]” [illegible]

3. [illegible] “[illegible]” [illegible]

[illegible] 2 [illegible] 1 [illegible] 1 [illegible] 2 [illegible] 1 [illegible] 2 [illegible] 1 [illegible] 2 [illegible] 2 [illegible] 1 [illegible] 2 [illegible] 1 [illegible]

[illegible] (A1) [illegible] (A2*) [illegible] A1 [illegible] A2 [illegible] 1 [illegible] 2 [illegible] A2* [illegible] T [illegible] 2 [illegible] 1 [illegible] 2 [illegible] T* [illegible] A1 [illegible] 2 [illegible] 1 [illegible]

[illegible]

```
template<typename T>
void t(T*, T const* = 0, ...);
template<typename T>
void t(T const*, T*, T* = 0);
void example(int* p)
{
    t(p, p);
}
```

[illegible]1[illegible]...[illegible]2[illegible]1[illegible]

[illegible](A1*, A1 const*)[illegible](A2 cosnt*, A2)[illegible]2[illegible](A1*, A1 const*)[illegible]A1 const[illegible]T[illegible](A1*, A1 const*)[illegible] t<A1 const>(A1 const*, A1 const*, A1 const* = 0)[illegible]const[illegible]1[illegible]A2 const*, A2*[illegible]

[illegible]

12.2.4 [illegible]

[illegible]

```
// details/nontmpl.cpp
#include <string>
#include <iostream>
template<typename T>
```

```
std::string f(T)
{
      return "Template";
}
std::string f(int&)
{
      return "Nontemplate";
}
int main()
{
      int x = 7;
      std::cout << f(x) << std::endl;
}
```

[illegible]

```
Nontemplate
```

12.3 [illegible]

[illegible] C++ [illegible] “[illegible]” [illegible]

[illegible]

[illegible]

12.3.1 [illegible]

[illegible]3[illegible]template[illegible]< [illegible] > [59] [illegible]

```
template<typename T>
class S {
  public:
    void info() {
      std::cout << "generic (S<T>::info())\n";
    }
};
template<>
class S<void> {
  public:
    void msg() {
      std::cout << "fully specialized (S<void>::msg())\n";
    }
};
```

[illegible]info[illegible]msg[illegible]

[illegible]

```
template<typename T>
class Types {
```

```
    public:
      typedef int I;
  };
  template<typename    T,    typename    U    =    typename
Types<T>::I>
  class S;                              // (1)
  template<>
  class S<void> {                       // (2)
    public:
      void f();
  };
  template<> class S<char, char>; // (3)
  template<> class S<char, 0>;    // □□□□□□0□□□U
  int main()
  {
    S<int>*       pi;    // □□□□□□1□□□□□□□□□
    S<int>         e1;    // □□:□□□1□□□□□□□□□□□□□
    S<void>*      pv;    // □□□□□□2□
    S<void,int>   sv;    // □□□□□□2□□□□□□□□□□
    S<void,char> e2;    // □□□□□□1□,□□□□□□□□□□□
    S<char,char> e3;    // □□□□□□3□□□□□□□□□□□□□
  }
  template<>
  class S<char, char> {   // (3)□□□□
  };
```

□□□
□□□

[illegible]“[illegible]”[illegible]

[illegible]template<>[illegible]

```
template<typename T>
class S;
template<> class S<char**> {
   public:
      void print() const;
};
//[illegible]template<>[illegible]
void S<char**>::print() const
{
      std::cout << "pointer to pointer to char\n";
}
```

[illegible]

```
template<typename T>
class Outside {
   public:
      template<typename U>
      class Inside {
      };
};
template<>
class Outside<void> {
   // [illegible]
```

```
    template<typename U>
    class Inside {
        private:
        static int count;
    };
};
//[illegible]template<>[illegible]
template<typename U>
int Outside<void>::Inside<U>::count = 1;
```

[illegible]

```
template <typename T>
class Invalid {
};
Invalid<double> x1;    // [illegible]Invalid<double>[illegible]
template<>
class Invalid<double>; // [illegible]Invalid<double>[illegible]
```

[illegible]C++[illegible]

```
// [illegible]1:
template<typename T>
class Danger {
    public:
        enum { max = 10 };
};
```

```
char buffer[Danger<void>::max];   // [illegible]extern void
clear(char const*);
int main()
{
    clear(buffer);
}
// [illegible]2:
template<typename T>
class Danger;
template<>
class Danger<void> {
    public:
      enum { max = 100 };
};
void clear(char const* buf)
{
    // [illegible]
    for(intk=0;k<Danger<void>::max; ++k) {
      buf[k] = '\0';
    }
}
```

[illegible]“[illegible]”[illegible]

12.3.2 [illegible]

[illegible]

[illegible]

```
template<typename T>
int f(T)                    // (1)
{
    return 1;
}
template<typename T>
int f(T*)                   // (2)
{
    return 2;
}
template<> int f(int)    // OK: [illegible]1[illegible]
{
    return 3;
}
template<> int f(int*)   // OK: (2)[illegible]
{
    return 4;
}
```

[illegible]

```
template<typename T>
int f(T, T x = 42)
```

```
{
    return x;
}
template<> int f(int, int = 35)   // [illegible]
{
    return 0;
}
template<typename T>
int g(T, T x = 42)
{
    return x;
}
template<> int g(int, int y)
{
    return y/2;
}
int main()
{
    std::cout << g(0) << std::endl; // [illegible]21
}
```

[illegible] g [illegible]

```
#ifndef TEMPLATE_G_HPP
#define TEMPLATE_G_HPP
```

```
// [illegible]:
template<typename T>
int g(T, T x = 42)
{
    return x;
}
// [illegible]
//[illegible]
template<> int g(int, int y);
#endif // TEMPLATE_G_HPP
```

The corresponding implementation file may read:

```
#include "template_g.hpp"
template<> int g(int, int y)
{
    return y/2;
}
```

[illegible]

12.3.3 [illegible]

[illegible]template<>[illegible]template<>[illegible]

```
template<typename T>
class Outer {                                   // (1)
  public:
    template<typename U>
```

```
        class Inner {                                   // (2)
            private:
        };
            static int count;                      // (3)
        static int code;                            // (4)
        void print() const {                      // (5)
            std::cout << "generic";
        }
    };
    template<typename T>
    int Outer<T>::code = 6;                        // (6)
    template<typename T> template<typename U>
    int Outer<T>::Inner<U>::count = 7;       // (7)
    template<>
    class Outer<bool> {                            // (8)
        public:
        template<typename U>
        class Inner {                                   // (9)
            private:
                static int count;                     // (10)
        };
        void print() const {                      // (11)
        }
    };
```

[illegible]1[illegible]Outer[illegible]4[illegible]code[illegible]5[illegible]print()[illegible] template<>[illegible]

```
template<>
int Outer<void>::code = 12;
template<>
void Outer<void>::print() const
{
    std::cout << "Outer<void>";
}
```

[illegible]Outer<void>[illegible]4[illegible]5[illegible]
Outer<void>[illegible]1[illegible]
[illegible]Outer<void>[illegible]

[illegible]C++[illegible]

```
template<>
int Outer<void>::code;
template<>
void Outer<void>::print() const;
```

[illegible]Outer<void>::code[illegible]

[illegible]

```
class DefaultInitOnly {
  public:
    DefaultInitOnly() {
    }
  private:
```

```
        DefaultInitOnly(DefaultInitOnly const&);   // □□□□□□□
};
template<typename T>
class Statics {
  private:
    static T sm;
};
// □□□□□□□□
// □□□□□□□□□□□□□□□□
template<>
DefaultInitOnly Statics<DefaultInitOnly>::sm;
```

□□□□□□Outer<T>::Inner□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ Outer<T>□□□□□□□□□□□□□□ Outer<T>□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Outer<T>□□□□□□□□□□□□template<>□□□□□□□□□□□□□□□□

```
template<>
  template<typename X>
  class Outer<wchar_t>::Inner {
    public:
      static long count; // □□□□□□□□□
  };
template<>
  template<typename X>
  long Outer<wchar_t>::Inner<X>::count;
```

□□Outer<T>::Inner□□□□□□□□□□□□□□Outer<T>□□□□□□□□□□□□□□□□□□□ template<>□□□□□□□□□□□□ template<>□□□□□□□□□□□□□□□□□inner template□□□□□□template<>□□□

```
template<>
   template<>
   class Outer<char>::Inner<wchar_t> {
      public:
         enum { count = 1};
   };
// [illegible]C++[illegible]:
// template<> [illegible]
template<typename X>
template<> class Outer<X>::Inner<void>; // [illegible]
```

[illegible]Outer<bool>[illegible]Outer<bool>[illegible]template<>[illegible]

```
template<>
class Outer<bool>::Inner<wchar_t> {
   public:
      enum { count = 2 };
};
```

12.4 [illegible]

[illegible]“[illegible]”[illegible]“[illegible]”[illegible]

```
template<typename T>
class List {              // (1)
   public:
      ...
```

```
        void append(T const&);
        inline size_t length() const;
        ...
    };
```

□□□List<T>::append()□□□□□□□□□□□□□□□□□□□□□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ List<int*>::append() □ □ □ □ List<void*>::append()□□□□□□□□□□□□□□□□□□□□□□□List□□□□□□□□□□□□□□□□C++□□□□□□□□□□□□□□□□□□□□□List□□□□□□□□□□□□□□□□□□□□□□□□□

```
template<typename T>
class List<T*> { // (2)
    private:
        List<void*> impl;
        ...
    public:
        ...
        void append(T* p) {
            impl.append(p);
        }
        size_t length() const {
            return impl.length();
        }
        ...
};
```

□□□□□□□□□□□□□□□□□□□□□□1□□

[illegible]template<…>[illegible]<T*>[illegible]

[illegible] List<void*> [illegible] List<void*>[illegible]

```
template<>
class List<void*> {    // (3)
    …
    void append (void* p);
    inline size_t length() const;
    …
};
```

[illegible]List[illegible]List<void*>[illegible]C++[illegible]

[illegible]

1.[illegible]

2.[illegible]

3.[illegible]2*N[illegible]N[illegible]

4.[illegible]

[illegible]

```
template<typename T, int I = 3>
class S;                            // [illegible]
template<typename T>
class S<int, T>;                // [illegible]
template<typename T = int>
class S<T, 10>;                  // [illegible]
template<int I>
```

```
class S<int, I*2>;          // [illegible]
template<typename U, int K>
class S<U, K>;               // [illegible]
                             //     [illegible]
```

[illegible]“[illegible]”[illegible]“[illegible]”[illegible]

[illegible]List[illegible]1[illegible]List[illegible]（pointer-to-member-pointers）[illegible]

```
template<typename C>
class List<void* C::*> {   // (4)
  public:
    // [illegible]void*[illegible]
    // [illegible]void*[illegible]
    typedef void* C::*ElementType;
    …
    void append(ElementType pm);
    inline size_t length() const;
    …
};
template<typename T, typename C>
class List<T* C::*> {      // (5)
  private:
    List<void* C::*> impl;
```

```
    ...
  public:
    // [illegible]
    // [illegible]void*[illegible]
    // [illegible]
    // [illegible]
    typedef T* C::*ElementType;
    ...
    void append(ElementType pm) {
      impl.append((void* C::*)pm);
    }
    inline size_t length() const {
      return impl.length();
    }
    ...
};
```

[illegible]4[illegible]5[illegible]4[illegible]5[illegible]

12.5 [illegible]

[illegible]C++[illegible]C++[illegible]HP[illegible]C++[illegible]EDG[illegible]C++ [illegible]Steve Adamczyk[illegible]John Spicer[illegible]EDG[illegible]

[illegible]12.4[illegible] List<T*>[illegible]Erwin Unruh [illegible]template metaprogramming[illegible]metaprogramming[illegible]17[illegible]

[illegible]13[illegible]

[illegible]

```
// [illegible]
template<typename T1, typename T2> class Pair;
template<int N1, int N2> class Pair;
```

[illegible]

[illegible]13[illegible]

[illegible]1988[illegible]1998[illegible]C++[illegible]1997[illegible]12[illegible]C++[illegible]C++[illegible]idioms[illegible]idioms[illegible]

[illegible]C++[illegible]C++[illegible]C++[illegible]C++[illegible]C++[illegible]

13.1 [illegible]Hack

[illegible]

```
#include <list>
#include <vector>
typedef std::vector<std::list<int> > LineTable;    // [illegible]
typedef std::vector<std::list<int>>    OtherTable;   // [illegible]
```

[illegible] 2 [illegible] typedef [illegible]“[illegible]>>[illegible]”[illegible]

[illegible] C++[illegible]“[illegible]>>[illegible] hack[illegible]”[illegible]

[illegible]C++[illegible] OtherTable [illegible]hack[illegible]>>[illegible]

```
template<int N> class Buf;
template<typename T> void strange() {}
template<int N> void strange() {}
int main()
{
    strange<Buf<16>>2> >(); // [illegible]>>[illegible]
}
```

[illegible] <: [illegible]9.3.1[illegible]

```
template<typename T> class List;
```

```
class Marker;
List<::Marker>* markers; // [illegible]
```

[illegible]List[:Marker>* markers[illegible]List[illegible]<:[illegible]

13.2 [illegible]typename[illegible]

[illegible]typename[illegible]5.1[illegible]9.3.2[illegible]Array<T>::ElementT [illegible] typename [illegible]Array<int>::ElementT[illegible]typename[illegible]

```
template <typename T>
class Array {
    public:
    typedef T ElementT;
    ...
};
template <typename T>
void clear (typename Array<T>::ElementT& p);      // [illegible]
void clear (typename Array<int>::ElementT& p);   // [illegible]
template<>
```

[illegible]C++[illegible]“[illegible]”[illegible]struct[illegible]class[illegible]union[illegible]enum[illegible]typename[illegible].template[illegible]->template[illegible]::template[illegible]3[illegible]

[illegible]typename[illegible]template[illegible]

[illegible]clear()[illegible]Array<T>::ElementT[illegible]typename[illegible]C++[illegible]typename[illegible]template[illegible]

13.3 [illegible]

[illegible]C++[illegible]

[illegible]

```
template <typename T1, typename T2 = int>
T2 count (T1 const& x);
class MyInt {
    …
};
void test (Container const& c)
{
    int i = count(c);
    MyInt j = count<MyInt>(c);
    assert(j == i);
}
```

[illegible]

```
template <typename T1 = int, typename T2>
class Bad;
```

```
Bad<int>* b;   // int[illegible]T1[illegible]T2[illegible]
```

[illegible]

```
template <typename T1 = int, typename T2>
T1 count (T2 const& x);
void test (Container const& c)
{
    int i = count(c);
    MyInt j = count<MyInt>(c);
    assert(j == i);
}
```

[illegible]C++[illegible]

[illegible]

```
template <typename T = double>
void f(T const& = T());
int main()
{
    f(1);            // [illegible]T[illegible]int
    f<long>(2);        // [illegible]T = long[illegible]
    f<char>();         // [illegible] f<char>('\0')
    f();             // [illegible] f<double>(0.0)
}
```

[illegible]

13.4 [illegible]

[illegible]

[illegible]

```
template <char const* msg>
class Diagnoser {
  public:
    void print();
};
int main()
{
    Diagnoser<"Surprise!">().print();
}
```

[illegible] C++[illegible] Diagnoser [illegible] Diagnoser<“X”>[illegible]Diagnoser<“X”>[illegible]“X”[illegible]char const[2][illegible]decay[illegible]char const*[illegible]

[illegible]C++[illegible]C++[illegible]

[illegible]

```
template <char const* str>
class Bracket {
```

```
   public:
      static char const* address();
      static char const* bytes();
};
template <char const* str>
char const* Bracket<str>::address()
{
   return str;
}
template <char const* str>
char const* Bracket<str>::bytes()
{
      return str;
}
```

[illegible]——[illegible]Brack<“X”>[illegible]“[illegible]C++[illegible]”[illegible]

[illegible]

```
template <double Ratio>
class Converter {
   public:
      static double convert (double val) {
         return val*Ratio;
      }
};
typedef Converter<0.0254> InchToMeter;
```

[illegible]C++[illegible]
[illegible]

13.5 [illegible]

[illegible]
[illegible]

```
#include <list>
   // std::list[illegible]:
   //   namespace std {
   //       template <typename T,
   //                         typename Allocator = allocator<T> >
   //       class list;
   // }
template<typename T1,
      typename T2,
         template<typename> class Container>
            // Container[illegible]
class Relation {
   public:
      …
   private:
      Container<T1> dom1;
      Container<T2> dom2;
};
int main()
{
```

```
    Relation<int, double, std::list> rel;
        // std::list
    ...
}
```

Container

std::list

std::list Container

std::list Container std::list

8.3.4

“ ”

C++ typedef

main()

```
template <typename T>
typedef std::list<T> MyList;
int main()
{
    Relation<int, double, MyList> rel;
}
```

typedef Container

C++

13.6 typedef

[illegible]typedef[illegible]

[illegible]C++[illegible]

```
template <typename T>
typedef vector<list<T> > Table;
```

[illegible]Table[illegible]typedef[illegible]

```
Table<int> t;             //t[illegible]vector<list<int> >
```

[illegible]typedef[illegible]typedef[illegible]

```
template <typename T>
class Table {
   public:
      typedef vector<list<T> > Type;
};
Table<int>::Type t; // t[illegible]vector<list<int> >
```

[illegible] typedef [illegible]

```
// [illegible] typedef [illegible]:
template<typename T> typedef T Opaque;
// [illegible]:
template<typename T> typedef void* Opaque<T*>;
// [illegible]:
template<> typedef bool Opaque<void>;
```

[illegible]typedef[illegible]typedef[illegible]

```
void candidate(long);
template<typename T> typedef T DT;
```

```
template<typename T> void candidate(DT<T>);
int main()
{
    candidate(42);   // □□□□ candidate()
}
```

□□□□□□□□□□□□□□□□□typedef□□□□□□□

13.7 □□□□□□□

□□ 12 □□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□

□□□“□□□□□□□□□□□□□□□□”□□□

□□

•□□C++□□□□□□std□□□□□□□□□□□□□std□□□□□□□□

•□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□R convert(T const&)□□R□T□□□□□□□□□R = void□□□□□□□□□□□□□□□

•□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□f(T)□g(T)□□T□□□□□□□g(&f<int>)□□□□□□□f□□

[illegible]f[illegible]

[illegible]“[illegible]”[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

```
template <typename T>
T const& max (T const&, T const&);          // [illegible]
template <typename T>
T* const& max <T*>(T* const&, T* const&); // [illegible]
```

[illegible]

```
template <typename T>
void add (T& x, int i);   // [illegible]
template <typename T1, typename T2>
void add (T1 a, T2 b);    // [illegible]
template <typename T>
void add<T*> (T*&, int); // [illegible]?
```

[illegible]

[illegible]C++[illegible]

13.8 typeof[illegible]

[illegible]

[illegible]

```
template <typename T1, typename T2>
```

```
Array<???> operator+ (Array<T1> const& x, Array<T2> const& y);
```

[illegible]x[illegible]y [illegible] x[0] + y[0][illegible]C++[illegible]T1[illegible]T2[illegible]

[illegible]typeof[illegible]sizeof [illegible]typeof[illegible]typeof[illegible]

```
template <typename T1, typename T2>
Array<typeof(T1()+T2())> operator+ (Array<T1> const& x,
                                    Array<T2> const& y);
```

[illegible]

```
template <typename T>
T makeT(); // [illegible]
template <typename T1, typename T2>
Array<typeof(makeT<T1>()+makeT<T2>())>
  operator+ (Array<T1> const& x,
             Array<T2> const& y);
```

[illegible]x[0][illegible]y[0][illegible]typeof[illegible]typeof[illegible]x[illegible]y[illegible]

```
// [illegible]:
template <typename T1, typename T2>
operator+ (Array<T1> const& x, Array<T2> const& y)
```

```
    -> Array<typeof(x[0]+y[0])>;
// □□□□□□□:
template <typename T1, typename T2>
function exp(Array<T1> const& x, Array<T2> const& y)
    -> Array<typeof(exp(x[0], y[0]))>
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□function□□□□□□□□□□operator□□□□□□□□□

□□□□□typeof□□□□□□□□□□typeof□□□□□□□□□□□□□□

```
class Base {
    public:
        virtual Base* clone();
};
class Derived : public Base {
    public:
    virtual Derived* clone(); // □□□□□
};
void demo (Base* p, Base* q)
{
    typeof(p->clone()) tmp = p->clone();
                            // tmp□□□□Base*
    ...
}
```

15.2.4□□□□□□typeof□□□□□□□□□□□promotion□□□trait□□□□□□□

13.9 □□□□

16.1[illegible]

[illegible]C++[illegible]Roland Hartinger[illegible] keyword argument [illegible] [StroustrupDnE][illegible] 6.5.1[illegible]

[illegible]

```
template<typename T,
      Move: typename M = defaultMove<T>,
      Copy: typename C = defaultCopy<T>,
      Swap: typename S = defaultSwap<T>,
      Init: typename I = defaultInit<T>,
      Kill: typename K = defaultKill<T> >
class Mutator {
   …
};
void test(MatrixList ml)
{
   mySort (ml, Mutator <Matrix, Swap: matrixSwap>);
}
```

[illegible] M[illegible]Move[illegible]

```
template<typename T,
      : typename Move = defaultMove<T>,
      : typename Copy = defaultCopy<T>,
```

```
        : typename Swap = defaultSwap<T>,
        : typename Init = defaultInit<T>,
        : typename Kill = defaultKill<T> >
class Mutator {
    ...
};
```

13.10 [illegible]

[illegible]15[illegible]19[illegible]trait[illegible]15.3.2[illegible] CSMtraits [illegible]

[illegible]“[illegible]”[illegible]type trait[illegible]C++[illegible]

```
#include <iostream>
int main()
{
        std::cout << std::type<int>::is_bit_copyable << '\n';
        std::cout << std::type<int>::is_union << '\n';
}
```

[illegible]C++[illegible]union[illegible]trait[illegible]“[illegible]”[illegible]CPU[illegible]

13.11 [illegible]

[illegible] C++[illegible]6.6.1 [illegible]

```
template <typename T>
void clear (T const& p)
{
   *p = 0; // [illegible] T [illegible]
}
template <typename T>
void core (T const& p)
{
   clear(p);
}
template <typename T>
void middle (typename T::Index p)
{
   core(p);
}
template <typename T>
void shell (T const& env)
{
   typename T::Index i;
   middle<T>(i);
}
class Client {
```

```
    public:
        typedef int Index;
        ...
};
Client main_client;
int main()
{
    shell(main_client);
}
```

[illegible] shell()[illegible]middle()[illegible] middle()[illegible] core()[illegible] shell()[illegible]int[illegible]core()[illegible] middle()[illegible]Client::Index[illegible]int[illegible]

[illegible]C++[illegible] [BCCL][illegible]shell()[illegible]

```
template <typename T>
void shell (T const& env)
{
    std::instantiation_error(
        !std::type<T>::has_member_type<"Index">,
        "T must have an Index member type");
    std::instantiation_error(
        !std::type<typename T::Index>::dereferencable,
```

```
        "T::Index must be a pointer-like type");
    typename T::Index i;
    middle(i);
}
```

[illegible]instantiation_error()[illegible]middle()[illegible]

[illegible]

```
template <typename T>
void shell (T const& env)
{
    template try {
        typename T::Index p;
        *p = 0;
    } catch "T::Index must be a pointer-like type";
        typename T::Index i;
        middle(i);
}
```

template try[illegible]

[illegible]C++[illegible]

13.12 [illegible]

[illegible]

```
template <typename T1>
class Tuple {
   // [illegible]
   ...
};
template <typename T1, typename T2>
class Tuple {
   // [illegible]
   ...
};
template <typename T1, typename T2, typename T3>
class Tuple {
   // 3[illegible]
   ...
};
```

[illegible]

[illegible]22[illegible]FunctionPtr[illegible]

```
template <typename T1, typename T2>
class Pair {
   //[illegible]
   ...
};
template <int I1, int I2>
```

```
class Pair {
   //[illegible]
   ...
};
```

[illegible]C++[illegible]

13.13 List[illegible]

[illegible]

[illegible]

```
#include <iostream>
template <typename T, ...list>
T const& max (T const&, T const&, list const&);
int main()
{
      std::cout << max(1, 2, 3, 4) << std::endl;
}
```

[illegible]list[illegible]

```
template <typename T> inline
T const& max (T const& a, T const& b)
{
      // [illegible]return a<b?b:a;
```

```
}
template <typename T, ...list> inline
T const& max (T const& a, T const& b, list const& x)
{
      return max (a, max(b,x));
}
```

□□□□□□□max(1,2,3,4)□□□□□□□□□□□□□4□□□□□□□□□□□□□max()□□□□□□□□□□□□□□T=int□list = int , int□□2□□□□□□□□□1□□□□1□□2□□□□max(2,3,4)□□□□□□max()□□□□□max(2,3,4)□□□□□□□□□max()□□□□□□□□□□□T = int□list = int□list□□□□□□□□□□□□□□ max(b,x)□□□□□□ max(3,4)□□□□□□□□□□□□□□

□□□list const&□□□□

□□□□□□□□□list□□□□□□□□□□□□□□□□□□□□□□□□□□□□ [ ]□□□□□□□□□□□□□□□□□□□□□□□□□□□□□metaprograming□□□□list□□□□□□□

```
template <typename T>
class ListProps {
   public:
      enum { length = 1 };
};
template <...list>
class ListProps {
   public:
      enum { length = 1+ListProps<list[1 ...]>::length };
};
```

[illegible] list [illegible]metaprogramming[illegible]

[illegible]list[illegible]

```
template <...list>
class Collection {
   list;
};
```

[illegible]list[illegible]Modern C++Design[illegible][AlexandrescuDesign][illegible]metaprogramming[illegible]

13.14 [illegible]

[illegible]“[illegible]T[illegible]”[illegible]T[illegible]“discriminated union”[illegible]variant type[illegible]tagged union[illegible]

```
template <...list>
class D_Union {
   public:
      enum { n_bytes };
      char bytes[n_bytes]; // [illegible]
                           // [illegible]
      ...
};
```

[illegible]n_bytes[illegible]sizeof(T)[illegible]T[illegible]bytes buffer[illegible] alignment requirement[illegible]alignment[illegible]

[illegible]“[illegible]alignment requirement[illegible]”[illegible] alignment [illegible] C[illegible]C++[illegible]_ _alignof_ _ [illegible]alignment[illegible]sizeof[illegible]alignment[illegible]sizeof[illegible]#pragma [illegible] alignment[illegible] alignof [illegible]alignment[illegible]alignment[illegible]

```
template <typename T>
class Alignment {
   public:
      enum { max = alignof(T) };
};
template <...list>
class Alignment {
   public:
      enum { max = alignof(list[0]) > Alignment<list[1 ...]>::max
                  ? alignof(list[0])
                  : Alignment<list[1 ...]>::max}
};
// [illegible]Size[illegible]
// [illegible]size
template <...list>
class Variant {
   public:
      char buffer[Size<list>::max] alignof(Alignment<list>::max);
```

```
    ...
};
```

13.15 类型检查和推导

有时候有必要用“类型检查和推导”来代替直接的类型声明。考虑下面这个例子。像下面这样的声明是否有点让人感到混乱？

```
std::map<std::string, std::list<int> >* dict
  = new std::map<std::string, std::list<int> >;
```

注意这里的类型名重复出现了两次，而且它又很长，因此许多程序员会使用typedef来简化这种声明。然而，还可以有一种方法，即根据被初始化的dict的表达式类型来推导出它的类型。也就是说，dict的类型可以由初始化时右侧的表达式来确定。例如，我们可以这样写：

```
dcl dict = new std::map<std::string, std::list<int> >;
```

在上面这个例子中，我们虚构了一个关键字用于声明变量dict，它的类型可以从初始化表达式推导出来。这里使用的dcl只是一个例子，其他语言中的var或者auto等关键字也可以起到类似的作用，在这里我们仅仅用于演示。

实际上，这个扩展的概念并不是新的，它可以追溯到早期的 Cfront 编译器（约 1982年）中。但是由于当时的一些限制，这个特性一直没有被包含进标准中，直到最近才开始被重新讨论。

另一个相关的问题是模板实参的推导。例如，我们可能会希望能够这样写：

```
std::list<> index = create_index();
```

或者当一个类模板的实参可以从构造函数实参中推导时，可以这样写：

```
template <typename T>
class Complex {
  public:
    Complex(T const& re, T const& im);...
};
Complex<> z(1.0, 3.0); // 推导 T = double
```

[illegible]Complex[illegible]

```
template <typename T>
class Complex {
  public:
    Complex(Complex<T> const&);
    template <typename T2> Complex(Complex<T2>
const&);…
};
Complex<double> j(0.0, 1.0);
Complex<> z = j; // [illegible]
```

[illegible]z[illegible]j[illegible]

13.16 [illegible]

[illegible] 22 [illegible]18[illegible]18.3[illegible]

[illegible]vector[illegible]vector[illegible]

```
class BigValue {
  public:
    void init();
  …
};
class Init {
  public:
```

```
        void operator() (BigValue& v) const {
            v.init();
        }
    };
    void compute (std::vector<BigValue>& vec)
    {
        std::for_each (vec.begin(), vec.end(),
                       Init());
        ...
    }
```

[illegible] Init[illegible]

[illegible]

```
    class BigValue {
        public:
            void init();
        ...
    };
    void compute (std::vector<BigValue>& vec)
    {
        std::for_each (vec.begin(), vec.end(),
                       $(BigValue&) { $1.init(); });
        ...
    }
```

[illegible]$ [illegible]

[illegible]$n[illegible]n[illegible]

[illegible]lambda[illegible]lambda[illegible]

closure[illegible]Java[illegible]

```
class BigValue {
  public:
    void init();
  ...
};
void compute (std::vector<BigValue>& vec)
{
  std::for_each (vec.begin(), vec.end(),
    class {
      public:
        void operator() (BigValue& v) const {
          v.init();
        }
    };
    );
  ...
}
```

[illegible]

[illegible]C++[illegible]Jaakko Järvi[illegible]Gary Powell[illegible]Lambda[illegible][LambdaLib][illegible]

13.17 [illegible]

[illegible]C++[illegible]1998[illegible]C++98[illegible]C++[illegible]
C++[illegible]

[illegible] C++[illegible]C++[illegible] ISO WG21/ANSI J16 [illegible]WG21/J16[illegible]C++0x[illegible]2001[illegible]4[illegible]
Copenhagen [illegible]C++0x[illegible]
WG21/J16[illegible]

[illegible]C++[illegible]C++[illegible]1990[illegible]STL[illegible]
C++[illegible]

[illegible]C++0x[illegible]C++98[illegible]C++[illegible]C++98[illegible]

---

[1]. [illegible]“[illegible]”[illegible]ordinary[illegible]“[illegible]”[illegible]“[illegible]”[illegible]“[illegible]”[illegible]namespace[illegible]“[illegible]”[illegible]“[illegible]”[illegible]“scope”[illegible]“[illegible]”[illegible]“[illegible]”[illegible]

[2]. [illegible]“their enclosing class”[illegible]Collection[illegible]Node[illegible]Handle[illegible]alloc()[illegible]

[3]. [illegible]built-in type[illegible]“[illegible]”[illegible]int[illegible]
double[illegible]“[illegible]”[illegible]

[4]. [illegible] T [illegible] 2 [illegible]

[5]. [illegible]

[6]. [illegible]“[illegible]”[illegible]

[7]. [illegible]class[illegible]class[illegible]typename[illegible]class[illegible]

[8]. [illegible]

[9]. [illegible]

[10]. [illegible]1[illegible]

[11]. [illegible]“[illegible]”[illegible]visible[illegible]9.2.2[illegible]10.1[illegible]

[12]. [illegible]“[illegible]”[illegible] qualified[illegible]

[13]. [illegible]“[illegible]”[illegible]“[illegible]”[illegible]

[14]. ADL[illegible]Koenig[illegible]Koenig[illegible]Andrew Koenig[illegible]

[15]. [illegible]

[16]. [illegible]

[17]. [illegible]VC6\VC7[illegible]2[illegible]f[illegible]2[illegible]f[illegible]ADL[illegible]main[illegible]using namespace N[illegible]f[illegible]namespace N[illegible]class N[illegible]using namespace X[illegible]using namespace N[illegible]using ::N[illegible]f[illegible]static[illegible]f[illegible]

[18]. [illegible]C++[illegible]

[19]. □□□□□□□□□tokenization□□□□□□□“□□”□“□□□□□lexing□”□□□
□□□□“□□□”□

[20]. □□□□□□□□□□□□□□□(Invert<1>)0 □□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□

[21]. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ # □[□]
□□□

[22]. □□[VandevoordeSolutions]□□□□□□□□□□□□C++□□□□□□□□□
□□□□□□□□□□□

[23]. □□□□□□□□□::□□□□□□□□□□□□□□□□□□□□□□

[24]. □□

[25]. □□□VC6□□□□□□□□

[26]. □□□□□□□□□□□two-phase lookup□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□1□□□□□□□□□□□□□□□□□2□□□□□10.3.1□□□□

[27]. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

[28]. □□□□□□□□□□□□□□□□□□□□□□

[29]. “□□□”□□□□□□□□□□□“□□□□□□□□□□”□□□□□□□□□□□□□□□□□□□
□□

[30]. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□7□□□□□□□□□
□□□□□□12□□□□□□□□□□□□

[31]. □□□union□□
□□□□□□□□□□□□□□□□□□□□

[32]. □□□□□□□□□□□□□□□□□□Block[0]□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□

[33]. □□□□□□□□□□□□□□union□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□

[34]. □□□□□□□□□□□□□□□□□□□std::auto_ptr<T>□□□□20□□

[35]. □□□two-phase lookup□□□□□□□□□two-stage lookup□□
two-phase name lookup□□□□□□□

[36]. □□□□□9.2.1□□“□□□”□□□

[37]. □□□□□10.3.5□□

[38]. □2002□□C++□□□□□□□□□□□□□□□□□□□□□□□□□□
typede□f □typedef int Int□□□□□□

[39]. □□□□□□□□□□□□□□□□□□□□□□□□□□

[40]. □□□□□□□□□□□□□□□□□□□□□□

[41]. □□□“□□□□□□□□”□□□instantiated member of template□□
□□□“□□□□□□”□

[42]. □□□□□□□“□□□inline□□□□□□□□□”□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□

[43]. □□□□□□□□□□□□□□□□□□□□□□ C++□□□□□□□□□□□□□
[LippmanObjMod]□

[44]. □□□□□□□□Cfront□□□□□□□□□Cfront□□□□□□□□□□□□□
□□□C++□□□□□□□□□□□□□Cfront□Cfront□3.0□□□1991□□
□□□□□□□□□□□□□□□□□3.0.1□□□□□□□□□□□□□

[45]. □□□□□□□□1□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□

[46]. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

[47]. HP[illegible]C++[illegible]Taligent[illegible]IBM[illegible]HP[illegible]C++[illegible]

[48]. EDG[illegible]C++[illegible] C++[illegible]EDG [illegible]

[49]. [illegible]specialized[illegible]specialization[illegible]

[50]. [illegible]mangled name[illegible]

[51]. [illegible]SFINAE[illegible]8.3.1[illegible]

[52]. decay[illegible]

[53]. [illegible]“[illegible]”[illegible]

[54]. [illegible]12.2[illegible]“[illegible]C++[illegible]”[illegible]

[55]. [illegible]S[illegible]w[illegible]w[illegible]S[illegible]

[56]. [illegible]

[57]. [illegible]0[illegible]null[illegible]null[illegible]

[58]. [illegible]C++[illegible]

[59]. [illegible]C++[illegible]

□3□□ □□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□
□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□1□□□□□2□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□

- □□□□
- trait□
- policy class□
- metaprogramming□
- □□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□

□14□ □□□□□□□

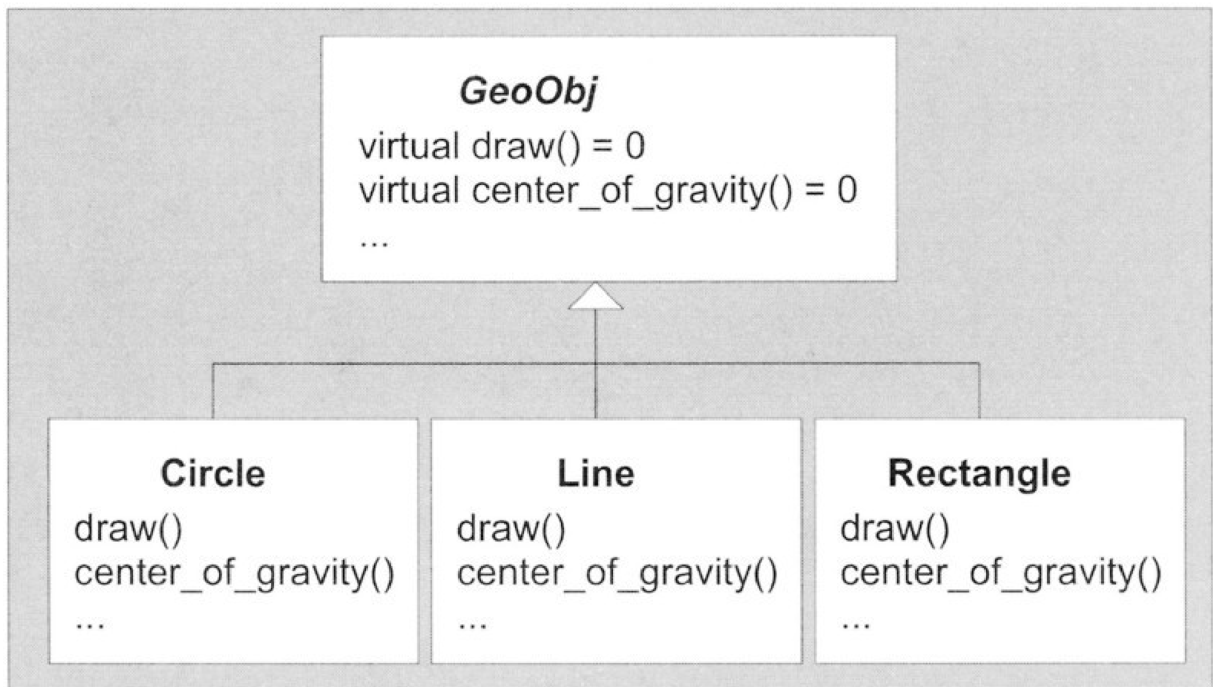
□□□□□□□□□□□□□□□□□□□□ [1] □□□□□□□□□□□□□□
□□□□□□□□C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□dynamic
polymorphism□□□□□□□□C++□□□□□□□□□□□□□□□□□□□□

□□
□□□□□□□□□□□□□static polymorphism□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

14.1 □□□

□C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□[2] □□

□□□abstract base class□ABC□GeoObj□□GeoObj□□□14.1□□



□14.1 □□□□□□□□□

```
// poly/dynahier.hpp
#include "coord.hpp"
```

```
// [illegible]GeoObj
class GeoObj {
   public:
      // [illegible]:
      virtual void draw() const = 0;
      // [illegible]:
      virtual Coord center_of_gravity() const = 0;
      …
};
// [illegible]Circle
// - [illegible]GeoObj
class Circle : public GeoObj {
   public:
      virtual void draw() const;
      virtual Coord center_of_gravity() const;…
};
// [illegible] Line
// - [illegible] GeoObj
class Line : public GeoObj {
   public:
   virtual void draw() const;
   virtual Coord center_of_gravity() const;
   …
};
…
```

[illegible]

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□

```
// poly/dynapoly.cpp
#include "dynahier.hpp"
#include <vector>
// □□□□□GeoObj
void myDraw (GeoObj const& obj)
{
   obj.draw();                  // □□□□□□□□□□□□□draw()
}
// □□□□ GeoObj□□□□□□□□□
Coord distance (GeoObj const& x1, GeoObj const& x2)
{
   Coord           c           =           x1.center_of_gravity()           -
x2.center_of_gravity();
   return c.abs();          // □□□□□□□□
}
// □□□□□□□□□GeoObj□□
void drawElems (std::vector<GeoObj*> const& elems)
{
   for (unsigned i=0; i<elems.size(); ++i) {
      elems[i]->draw();    // □□□□□□□□□□□□□ draw()
   }
}
int main()
{
   Line l;
```

```
Circle c, c1, c2;
myDraw(l);                     //  myDraw(GeoObj&)  =>
Line::draw()
myDraw(c);                     //  myDraw(GeoObj&)  =>
Circle::draw()
distance(c1,c2);      // distance(GeoObj&,GeoObj&)
distance(l,c);        // distance(GeoObj&,GeoObj&)
std::vector<GeoObj*> coll;  // □□□□□□□□
coll.push_back(&l);         // □□□□□□
coll.push_back(&c);         // □□□□□
drawElems(coll);            // □□□□□ GeoObj□□
}
```

□□□□□□□draw()□center_of_gravity()□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ mydraw()□distance()□drawElems()□□□□□□□□□ 3 □□□ mydraw()□□□□□□□□□□GeoObj □□□□□□□□□□□□□□□□□□□□□□□□□□□□□ draw()□center_of_gravity()□□ Line □□□mydraw()□□□□obj.draw()□□□□Line::draw()□□□□□□draw()□□□□□□Circle□□□□□□□□Circle::draw()□□□□□□□distance()□□□□□□□□□□□□□□□□center_of_gravity()□□□

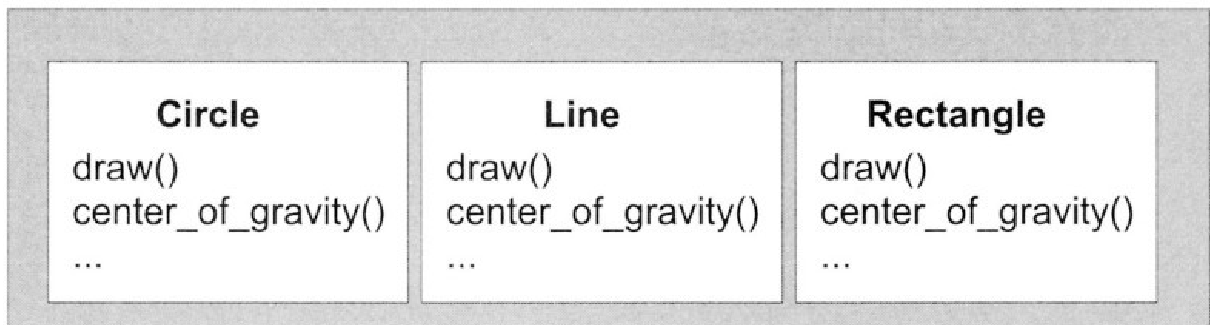
□□□□□□□□□□□□□□□□□□□□□□□□□□□□drawElems□□□□□□□□□□□□□□□□

elems[i]->draw()

□□□□□□□□□□□□□□□□□□□

14.2 [illegible]

[illegible] 14.2 [illegible]



[illegible]14.2 [illegible]

[illegible] myDraw() [illegible]

```
void myDraw (GeoObj const& obj)    // GeoObj [illegible]
{
  obj.draw();
}
```

[illegible]

```
template <typename GeoObj>
void myDraw (GeoObj const& obj)    // GeoObj [illegible]
{
  obj.draw();
}
```

[illegible] myDraw() [illegible] GeoObj [illegible]

[illegible]myDraw()[illegible]
myDraw<Line>()[illegible]myDraw<Circle>()[illegible]

[illegible]

```
// poly/statichier.hpp
#include "coord.hpp"
// [illegible] Circle
// - [illegible]
class Circle {
   public:
      void draw() const;
      Coord center_of_gravity() const;…
};
// [illegible]Line
// - [illegible]
class Line {
   public:
      void draw() const;
      Coord center_of_gravity() const;
      …
};
…
```

[illegible]:

```
// poly/staticpoly.cpp
#include "statichier.hpp"
#include <vector>
// [illegible] GeoObj
```

```
template <typename GeoObj>
void myDraw (GeoObj const& obj)
{
    obj.draw();    // □□□□□□□□□□□□draw()
}
// □□□□GeoObj□□□□□□□□□
template <typename GeoObj1, typename GeoObj2>
Coord distance (GeoObj1 const& x1, GeoObj2 const& x2)
{
    Coord c = x1.center_of_gravity() - x2.center_of_gravity();
    return c.abs(); // □□□□□□□□
}
// □□□□□□□□□GeoObj□□
template <typename GeoObj>
void drawElems (std::vector<GeoObj> const& elems)
{
    for (unsigned i=0; i<elems.size(); ++i) {
        elems[i].draw();    // □□□□□□□□□□□□draw()
    }
}
int main()
{
    Line l;
    Circle c, c1, c2;
    myDraw(l);          // myDraw<Line>(GeoObj&) => Line::draw()
```

```
    myDraw(c);              // myDraw<Circle>(GeoObj&) =>
Circle::draw()
    distance(c1,c2);
            //distance<Circle,Circle>(GeoObj1&,GeoObj2&)
    distance(l,c);                    //      distance<Line,Circle>
(GeoObj1&,GeoObj2&)
    // std::vector<GeoObj*> coll;   // [illegible]: [illegible]
    std::vector<Line> coll;   // [illegible]
    coll.push_back(l);        // [illegible]
    drawElems(coll);          // [illegible]
}
```

[illegible]distance()[illegible]myDraw()[illegible]GeoObj[illegible]GeoObj1[illegible]GeoObj2[illegible]

```
distance(l,c);          //          distance<Line,Circle>
(GeoObj1&,GeoObj2&)
```

[illegible]

14.3 [illegible]

[illegible]

14.3.1 [illegible]

[illegible]C++[illegible]idioms[illegible] [3] [illegible]

• [illegible]

[illegible]

[illegible]

• [illegible]

[illegible]

[illegible]

[illegible] C++ [illegible] Smalltalk [illegible] C++ [illegible]

14.3.2 [illegible]

C++ [illegible]

• [illegible]

• [illegible]

• [illegible]

[illegible] C++ [illegible]

• [illegible]

• [illegible]

• [illegible]

[illegible]

[illegible]

[illegible] + [illegible]

14.3.3 [illegible]

[illegible]

[illegible]16[illegible]16[illegible]cuiriously recurring template pattern[illegible]CRTP[illegible]

14.4 [illegible]

[illegible]C++[illegible]bridge pattern[illegible][DesignPatternsGoV][illegible]14.3[illegible]

Interface

Implementation* body;

operationA() {
body->operationA()
}
operationB() {
body->operationB()
body->operationC()
}

body

Implementation

virtual operationA() = 0;
virtual operationB() = 0;
virtual operationC() = 0;

Implementation A

virtual operationA();
virtual operationB();
virtual operationC();

Implementation B

virtual operationA();
virtual operationB();
virtual operationC();

[illegible]14.3 [illegible]

[illegible]14.4[illegible]

Impl

Interface

Impl body;

operationA() {
body.operationA()
}
operationB() {
body.operationB()
body.operationC()
}

Implementation A

operationA();
operationB();
operationC();

Implementation B

operationA();
operationB();
operationC();

[illegible]14.4 [illegible]

14.5 [illegible]

[illegible][CzarneckiEiseneckerGenPorg][illegible]

[illegible]……[illegible]

169[illegible]170[illegible]

[illegible]C++[illegible]C++[illegible]

[illegible]C++[illegible]STL[illegible]Standard Template Library[illegible] C++[illegible]STL [illegible]STL[illegible]

[illegible]

```
template <class Iterator>
Iterator max_element (Iterator beg,    // [illegible]
                      Iterator end)    // [illegible]
{
    // [illegible]
    // [illegible]
    // [illegible]Iterator[illegible]
    ...
}
```

[illegible]max_element()[illegible]

```
namespace std {
  template <class T, ... >
```

```
    class vector {
      public:
        typedef ... const_iterator;     // [illegible]vector[illegible]
        ...                             // [illegible]
        const_iterator begin() const; // [illegible]
        const_iterator end() const;   // [illegible]
        ...
    };
    template <class T, ...>
    class list {
      public:
        typedef ... const_iterator;     // [illegible]list[illegible]
        ...                             //   [illegible]
        const_iterator begin() const; // [illegible]
        const_iterator end() const;   // [illegible]
        ...
    };
}
```

[illegible]max_element()[illegible]

```
// poly/printmax.cpp
#include <vector>
#include <list>
#include <algorithm>
#include <iostream>
#include "MyClass.hpp"
template <typename T>
```

```
void print_max (T const& coll)
{
   // [illegible]
   typename T::const_iterator pos;
   // [illegible]
   pos = std::max_element(coll.begin(),coll.end());
   //[illegible]coll[illegible]
   if (pos != coll.end()) {
      std::cout << *pos << std::endl;
   }
   else {
      std::cout << "empty" << std::endl;
   }
}
int main()
{
   std::vector<MyClass> c1;
   std::list<MyClass> c2;
   …
   print_max (c1);
   print_max (c2);
}
```

STL[illegible]

[illegible]“[illegible]”[illegible]

concept [illegible] STL[illegible]

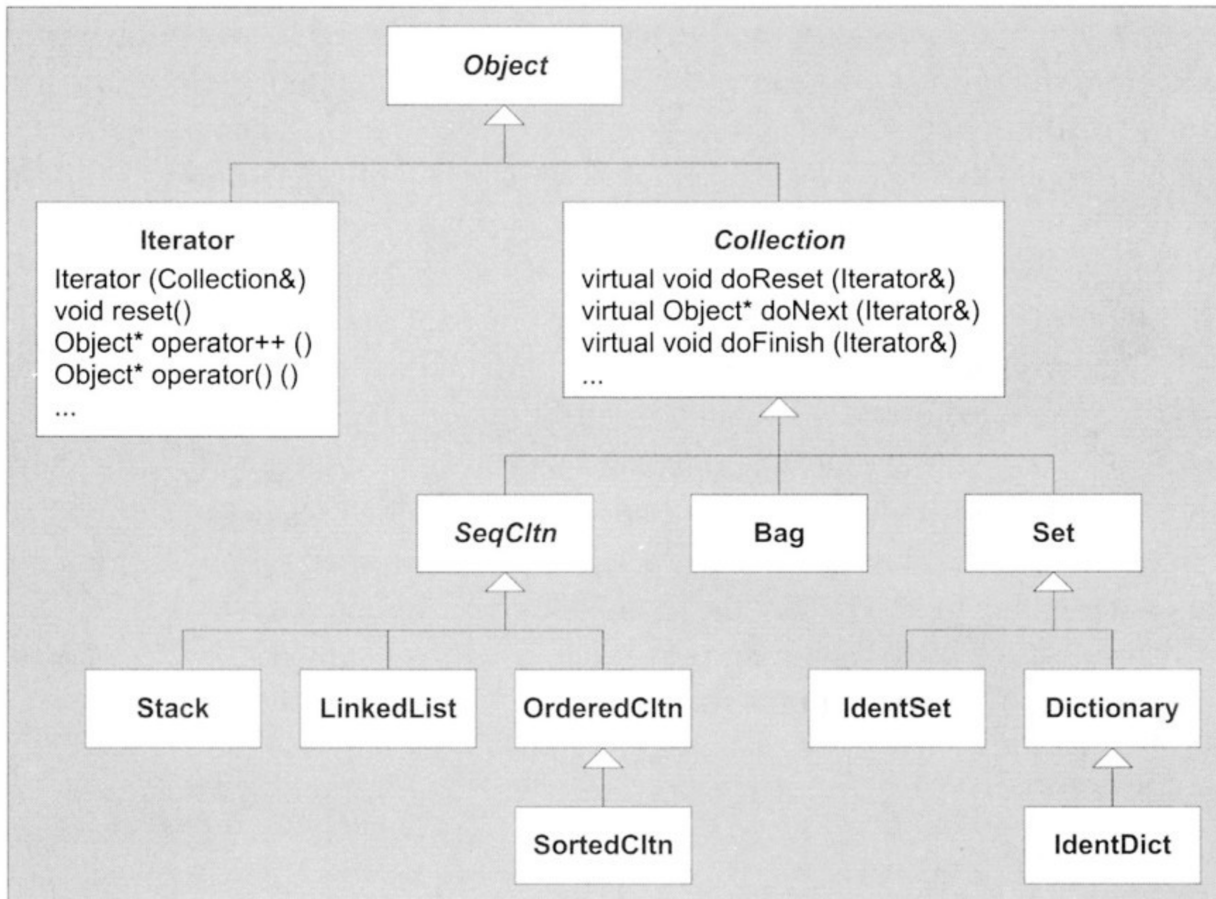
[illegible]

[illegible] STL [illegible]

[illegible]

14.6 [illegible]

[illegible]C++[illegible]National Institutes of Health Class Library [illegible]NIHCL[illegible]Smalltalk[illegible]14.5[illegible]



□14.5 NIHCL□□□□□

□□□ C++□□□□NIHCL□□□□□□□□□□□□□□□□□□□□□□□□□□□Smalltalk□□□Iterator□□□□□Collection□□□□□□□□□□

Bag c1;

Set c2;

...

Iterator i1(c1);

Iterator i2(c2);

...

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□C++□□□C++□□□

Smalltalk[illegible]NIHCL [illegible]

[illegible]

[illegible]

[illegible]C++[illegible]STL[illegible]

[illegible]C++[illegible]

[illegible]15[illegible] trait[illegible]policy[illegible]

[illegible]“[illegible]”[illegible]

[illegible]

policy[illegible]trait[illegible]trait[illegible]C++[illegible]

[illegible]

15.1 [illegible]

[illegible]（policy）[illegible]trait[illegible]

15.1.1 fixed traits

[illegible]1[illegible] [4] [illegible]

```
// traits/accum1.hpp
#ifndef ACCUM_HPP
#define ACCUM_HPP
template <typename T>
inline
T accum (T const* beg, T const* end)
{
   T total = T(); // [illegible]T()[illegible]0[illegible]
   while (beg != end) {
      total += *beg;
      ++beg;
   }
   return total;
}
#endif // ACCUM_HPP
```

[illegible]0[illegible]T()[illegible]int[illegible]float[illegible]T()[illegible]5.5[illegible]

[illegible]1[illegible]trait[illegible]accum()[illegible]

```
// traits/accum1.cpp
#include "accum1.hpp"
#include <iostream>
int main()
{
// [illegible]5[illegible]
int num[]={1,2,3,4,5};
// [illegible]
std::cout << "the average value of the integer values is "
      << accum(&num[0], &num[5]) / 5
      << '\n';
// [illegible]
char name[] = "templates";
int length = sizeof(name)-1;
// [illegible]
std::cout << "the average value of the characters in \""
      << name << "\" is "
      << accum(&name[0], &name[length]) / length
      << '\n';
}
```

[illegible]accum()[illegible]5[illegible]

```
int num[]={1,2,3,4,5};
```

```
...
accum(&num[0], &num[5])
```

[illegible]

[illegible]2[illegible]templates[illegible]a[illegible]z[illegible]ASCII[illegible]EBCDIC [5] [illegible]a[illegible]z[illegible]ASCII[illegible]a[illegible]97[illegible]z[illegible]122[illegible]97[illegible]122[illegible]

```
the average value of the integer values is 3
the average value of the characters in "templates" is -5
```

[illegible]char[illegible]char[illegible]AccT[illegible]AccT[illegible]total[illegible]

```
accum<int>(&name[0],&name[length])
```

[illegible]

[illegible]accum()[illegible]T[illegible]T[illegible]T[illegible]trait[illegible]

```
// traits/accumtraits2.hpp
template<typename T>
class AccumulationTraits;
template<>
class AccumulationTraits<char> {
  public:
    typedef int AccT;
```

```
};
template<>
class AccumulationTraits<short> {
  public:
    typedef int AccT;
};
template<>
class AccumulationTraits<int> {
  public:
    typedef long AccT;
};
template<>
class AccumulationTraits<unsigned int> {
  public:
    typedef unsigned long AccT;
};
template<>
class AccumulationTraits<float> {
  public:
    typedef double AccT;
};
```

[illegible] AccumulationTraits [illegible] trait [illegible]trait[illegible]trait[illegible]T[illegible]

[illegible]accum()[illegible]

```
// traits/accum2.hpp
```

```
#ifndef ACCUM_HPP
#define ACCUM_HPP
#include "accumtraits2.hpp"
template <typename T>
inline
typename AccumulationTraits<T>::AccT accum (T const* beg,
                                            T const* end)
{
  // □□□□□□□□□□trait
  typedef typename AccumulationTraits<T>::AccT AccT;
  AccT total = AccT(); // □□ AccT() □□□□□□0□
  while (beg != end) {
    total += *beg;
    ++beg;
  }
  return total;
}
#endif // ACCUM_HPP
```

□□□□□□□□□□□□□□□□□□□□

```
the average value of the integer values is 3
the average value of the characters in "templates" is 108
```

□□ accum() □□□□□□□□□ AccumulationTraits□□□□□□□□□□Acct□□□□□□□□□□□□□□□Acct□□□□□□□trait□□□□□□□□□□□□□□□□□□□

15.1.2 value trait

□□□□□□□□□□□□ trait □□□□□□“□”□□

□□□□□accum()□□□□□□□□□□□□□□□□□□□□□□□total□□□□□□□□□□□□□□□0□□□

```
AccT total = AccT(); // □□AccT() □□□□□□□0□
...
return total;
```

□□AccT□□□□□□□□□□□□□□

□ □ □ □ □ □ □ □ □ □ □ trait □ AccumulationTraits□□□□value trait□

```
// traits/accumtraits3.hpp
template<typename T>
class AccumulationTraits;
template<>
class AccumulationTraits<char> {
  public:
    typedef int AccT;
    static AccT const zero = 0;
};
template<>
class AccumulationTraits<short> {
  public:
    typedef int AccT;
    static AccT const zero = 0;
};
template<>
```

```
class AccumulationTraits<int> {
  public:
    typedef long AccT;
    static AccT const zero = 0;
};
...
```

□□□□□□□□□□□□trait□□□□□□□□□□□□□□□□□□□□□□□accum()□□□□□□□

```
// traits/accum3.hpp
#ifndef ACCUM_HPP
#define ACCUM_HPP
#include "accumtraits3.hpp"
template <typename T>
inline
typename AccumulationTraits<T>::AccT accum (T const* beg,
                                            T const* end)
{
  // □□□□□□□□□□trait
  typedef typename AccumulationTraits<T>::AccT AccT;
  AccT total = AccumulationTraits<T>::zero;
  while (beg != end) {
    total += *beg;
    ++beg;
  }
  return total;
}
```

```
#endif // ACCUM_HPP
```

[illegible]total[illegible]

```
AccT total = AccumulationTraits<T>::zero;
```

[illegible]C++[illegible]

```
...
template<>
class AccumulationTraits<float> {
public:
   typedef double AccT;
   static double const zero = 0.0;   // [illegible]
};
```

[illegible]value trait[illegible]

```
template<>
class AccumulationTraits<float> {
public:
   typedef double AccT;
   static double const zero;
};
```

[illegible]

```
...
double const AccumulationTraits<float>::zero = 0.0;
```

[illegible]zero[illegible]0[illegible]

[illegible]value trait[illegible]
[6] [illegible]AccumulationTraits[illegible]

```
// traits/accumtraits4.hpp
template<typename T>
class AccumulationTraits;
template<>
class AccumulationTraits<char> {
   public:
      typedef int AccT;
      static AccT zero() {
         return 0;
      }
};
template<>
class AccumulationTraits<short> {
   public:
      typedef int AccT;
      static AccT zero() {
         return 0;
      }
};
template<>
class AccumulationTraits<int> {
   public:
      typedef long AccT;
   static AccT zero() {
      return 0;
```

```
    }
};
template<>
class AccumulationTraits<unsigned int> {
  public:
    typedef unsigned long AccT;
    static AccT zero() {
      return 0;
    }
};
template<>
class AccumulationTraits<float> {
  public:
    typedef double AccT;
    static AccT zero() {
      return 0;
    }
};
…
```

[illegible]

```
AccT total = AccumulationTraits<T>::zero();
```

[illegible]trait [illegible]trait [illegible] accum()[illegible] accum()[illegible]trait[illegible]trait[illegible]

15.1.3 [illegible]trait

[illegible]trait[illegible]fixed trait[illegible]trait[illegible]trait[illegible]float[illegible]float[illegible]

[illegible]trait[illegible]trait[illegible] [7] [illegible]

[illegible]trait[illegible]

```
Accum<char>::accum(&name[0], &name[length])
```

[illegible]

```
// traits/accum5.hpp
#ifndef ACCUM_HPP
#define ACCUM_HPP
#include "accumtraits4.hpp"
template <typename T,
          typename AT = AccumulationTraits<T> >
class Accum {
  public:
    static typename AT::AccT accum (T const* beg, T const* end) {
          typename AT::AccT total = AT::zero();
          while (beg != end) {
             total += *beg;
             ++beg;
```

```
        }
        return total;
    }
};
#endif // ACCUM_HPP
```

[illegible]2[illegible]1[illegible]
[illegible]

[illegible]

```
template <typename T>
inline
typename AccumulationTraits<T>::AccT accum (T const* beg,
                                             T const* end)
{
    return Accum<T>::accum(beg, end);
}
template <typename Traits, typename T>
inline
typename Traits::AccT accum (T const* beg, T const* end)
{
    return Accum<T, Traits>::accum(beg, end);
}
```

15.1.4 policy[illegible]policy[illegible]

[illegible]accumulation[illegible]summation[illegible]
[illegible]

[illegible]
accum()[illegible]total +=*beg[illegible]
[illegible]policy[illegible]policy[illegible]
[illegible]policy [8] [illegible]

[illegible]Accum[illegible]

```
// traits/accum6.hpp
#ifndef ACCUM_HPP
#define ACCUM_HPP
#include "accumtraits4.hpp"
#include "sumpolicy1.hpp"
template <typename T,
          typename Policy = SumPolicy,
          typename Traits = AccumulationTraits<T> >
class Accum {
  public:
    typedef typename Traits::AccT AccT;
    static AccT accum (T const* beg, T const* end) {
      AccT total = Traits::zero();
      while (beg != end) {
        Policy::accumulate(total, *beg);
        ++beg;
    }
    return total;
  }
};
#endif // ACCUM_HPP
```

[illegible]SumPolicy[illegible]

```
// traits/sumpolicy1.hpp
#ifndef SUMPOLICY_HPP
#define SUMPOLICY_HPP
class SumPolicy {
  public:
    template<typename T1, typename T2>
    static void accumulate (T1& total, T2 const & value)
{
      total += value;
    }
};
#endif // SUMPOLICY_HPP
```

[illegible]policy[illegible]

[illegible] policy[illegible]

```
// traits/accum7.cpp
#include "accum6.hpp"
#include <iostream>
class MultPolicy {
  public:
    template<typename T1, typename T2>
    static void accumulate (T1& total, T2 const& value) {
      total *= value;
    }
};
int main()
```

```
{
    // □□□□□5□□□□□□
    int num[]={1,2,3,4,5};
    // □□□□□□□
    std::cout << "the product of the integer values is "
        <<      Accum<int,MultPolicy>::accum(&num[0],
    &num[5])
        << '\n';
}
```

□□□□□□□□□□□□□□□□

the product of the integer values is 0

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□0 □□□□□□□□□□□□□□□0□□□□□□□□□□□0□□□□□□□□□□□0□□□□□□□□□□□□□□trait□□□policy□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□policy□□□□□□□policy□□□□□zero()□trait□□□□□□□trait□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□trait□policy□□□□□□C++□□□accumulate()□□□□□□□□□□□□□□□□3□□□

15.1.5 trait□policy□□□□□

□□□□□□□□□□□□□□□□□□□□□□policy□□trait□□□□□□□□□□□□□trait□□□□□□□policy□□

New Shorter Oxford English Dictionary□□□□□□□□□□□

•trait n…□□□□□□□□□□□□□□□□□□□□

•policy n…□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□policy class□□ Andrei Alexandrescu□

Modern C++Design□□□□□□□□[AlexandrescuDesign]□□8□□□

policy□trait□□□□□□□□□□policy□□□□□□□□□trait□□□□□□□□□

□□□□□□□□trait□□□□1□□Nathan Myers□□□□□□□□□□□□□□□

trait class□□

□□□□□□□□□□□□□□“□□□□□□□□”□□□□□□□□□“□□□□□□□□”□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□

•trait□□□□□□□□□□□□□□□□□□

•policy □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□trait□□□□□□

•trait□□□fixed trait□□□□□□□□□□□□□□□□□□□□trait□□

•trait□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

•trait□□□□□□□□□□□□□□□□□□

•trait□□□□□trait□□□□□□□

□□policy class□□□□□□□□□□□□

•□□□□□□□□□□□□□□□□□□policy class□□□□□□□

•policy□□

□□□□□□policy□□

•policy□□□□□□□□□□□□□□□□□□□□□□□□□

•policy class□□□□□□□□□□□

•policy□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□C++□□□□□□

trait□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□trait□□□□□□□□□□□

□□□□□□□□□□□□[JosuttisStdLib]□11.2.14□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□trait□□□trait□□□□□□□□□□□policy□□□□□□

15.1.6 □□□□□□□□□□□□

[illegible]policy[illegible]SumPolicy[illegible]MutPolicy[illegible]policy class[illegible]policy class[illegible]SumPolicy[illegible]

```
// traits/sumpolicy2.hpp
#ifndef SUMPOLICY_HPP
#define SUMPOLICY_HPP
template <typename T1, typename T2>
class SumPolicy {
  public:
    static void accumulate (T1& total, T2 const & value)
{
      total += value;
    }
};
#endif // SUMPOLICY_HPP
```

[illegible]Accum[illegible]

```
// traits/accum8.hpp
#ifndef ACCUM_HPP
#define ACCUM_HPP
#include "accumtraits4.hpp"
#include "sumpolicy2.hpp"
template <typename T,
      template<typename,typename> class Policy =
  SumPolicy,
      typename Traits = AccumulationTraits<T> >
class Accum {
  public:
```

```
        typedef typename Traits::AccT AccT;
        static AccT accum (T const* beg, T const* end) {
            AccT total = Traits::zero();
            while (beg != end) {
                Policy<AccT,T>::accumulate(total, *beg);
                ++beg;
            }
            return total;
        }
    };
    #endif // ACCUM_HPP
```

[illegible]trait[illegible]AccT[illegible]policy[illegible]trait[illegible]trait[illegible]trait[illegible]

[illegible]policy class[illegible]policy class[illegible]1[illegible]

[illegible]policy [illegible]policy[illegible]SumPolicy[illegible]Boolean[illegible] += [illegible] + [illegible]15.1.4[illegible]SumPolicy[illegible]

```
// traits/sumpolicy3.hpp
#ifndef SUMPOLICY_HPP
#define SUMPOLICY_HPP
template<bool use_compound_op = true>
```

```
class SumPolicy {
  public:
    template<typename T1, typename T2>
    static void accumulate (T1& total, T2 const & value)
{
      total += value;
    }
};
template<>
class SumPolicy<false> {
  public:
    template<typename T1, typename T2>
    static void accumulate (T1& total, T2 const & value)
{
      total = total + value;
    }
};
#endif // SUMPOLICY_HPP
```

[illegible]Accum[illegible]

15.1.7 [illegible]policie [illegible]/[illegible] trait

[illegible]trait[illegible]policy[illegible]trait[illegible]policy[illegible]

[illegible]trait[illegible]policy[illegible]policy[illegible]

[illegible]16.1[illegible]13[illegible]

15.1.8 [illegible]

[illegible]trait[illegible]policy[illegible]accum()[illegible]accum()[illegible]accum()[illegible]C++[illegible]iterator trait[illegible]trait[illegible]accum()[illegible]

```
// traits/accum0.hpp
#ifndef ACCUM_HPP
#define ACCUM_HPP
#include <iterator>
template <typename Iter>
inline
typename std::iterator_traits<Iter>::value_type
accum (Iter start, Iter end)
{
    typedef                                                    typename
  std::iterator_traits<Iter>::value_type VT;
    VT total = VT(); // [illegible]VT()[illegible]0[illegible]
    while (start != end) {
        total += *start;
        ++start;
    }
    return total;
```

```
}
#endif // ACCUM_HPP
```

iterator_trait[illegible]

[illegible] trait[illegible]

```
namespace std {
    template <typename T>struct iterator_traits<T*> {
        typedef T                         value_type;
        typedef ptrdiff_t                 difference_type;
        typedef random_access_iterator_tag iterator_category;
        typedef T*                        pointer;
        typedef T&                        reference;
    };
}
```

[illegible] AccumulationTraits[illegible]

15.2 [illegible]

[illegible]trait[illegible] C [illegible] C++[illegible]value function[illegible]type function[illegible]

sizeof [illegible]sizeof[illegible]

```
// traits/sizeof.cpp
```

```
#include <stddef.h>
#include <iostream>
template <typename T>
class TypeSize {
  public:
    static size_t const value = sizeof(T);
};
int main()
{
  std::cout << "TypeSize<int>::value = "
    << TypeSize<int>::value << std::endl;
}
```

[illegible]trait[illegible]

15.2.1 [illegible]

[illegible] vector<T>[illegible]list<T>[illegible] stack<T>[illegible]

```
// traits/elementtype.cpp
#include <vector>
#include <list>
#include <stack>
#include <iostream>
#include <typeinfo>
template <typename T>
class ElementT;                          // [illegible]
template <typename T>
```

```
class ElementT<std::vector<T> > {   // □□□□
   public:
      typedef T Type;
};
template <typename T>
class ElementT<std::list<T> > {    // □□□□
   public:
      typedef T Type;
};
template <typename T>
class ElementT<std::stack<T> > {    // □□□□
   public:
      typedef T Type;
};
template <typename T>
void print_element_type (T const & c)
{
   std::cout << "Container of "
         << typeid(typename ElementT<T>::Type).name()
         << " elements.\n";
}
int main()
{
   std::stack<bool> s;
   print_element_type(s);
}
```

[illegible]value_type[illegible]

```
template <typename C>
class ElementT {
   public:
      typedef typename C::value_type Type;
};
```

[illegible]value_type[illegible]

```
template <typename T1, typename T2, ...>
class X {
   public:
      typedef T1 … ;
      typedef T2 … ;
      …
};
```

[illegible]

```
template <typename T, typename C>
T sum_of_elements (C const& c);
```

[illegible]sum_of_elements<int>(list)[illegible]

```
template<typename C>
```

typename ElementT<C>::Type sum_of_elements (C const& c);

□□□□□□□□□□□□□□□□□□□□□

□□□□□trait □□

□□□□□□□□□ElementT□□□trait class□□□□□□□□□□□□□□□□□C□□□trait□□□□□□□□□□□□□□trait□□□□□ElementT□trait class□□□□□□trait class□□□□□□□□□□□□□□□□□□□□□□□□“□□□”□□□□□□□□□□□□□□□□□□□□

15.2.2 □□class□□

□□□□□□□□□□□□□□□□□□□□□□□class□□□

```
// traits/isclasst.hpp
template<typename T>
class IsClassT {
  private:
    typedef char One;
    typedef struct { char a[2]; } Two;
    template<typename C> static One test(int C::*);
    template<typename C> static Two test(…);
  public:
    enum { Yes = sizeof(IsClassT<T>::test<T>(0)) == 1 };
    enum { No = !Yes };
};
```

□□□□□□□□□8.3.1□□□SFINAE□□□substitution-failure-is-not-error□□□□□□□□□□□□□□□SFINAE□□□□□□□□□□□□□□□□□□□□□class□

[illegible]C[illegible]class[illegible]C::*[illegible]

[illegible]class[illegible]

```
// traits/isclasst.cpp
#include <iostream>
#include "isclasst.hpp"
class MyClass {
};
struct MyStruct {
};
union MyUnion {
};
void myfunc()
{
}
enum E{e1}e;
// [illegible]
template <typename T>
void check()
{
   if (IsClassT<T>::Yes) {
      std::cout << " IsClassT " << std::endl;
   }
   else {
      std::cout << " !IsClassT " << std::endl;
   }
}
```

```
// [illegible]
template <typename T>
void checkT (T)
{
   check<T>();
}
int main()
{
   std::cout << "int: ";
   check<int>();
   std::cout << "MyClass: ";
   check<MyClass>();
   std::cout << "MyStruct:";
   MyStruct s;
   checkT(s);
   std::cout << "MyUnion: ";
   check<MyUnion>();
   std::cout << "enum: ";
   checkT(e);
   std::cout << "myfunc():";
   checkT(myfunc);
}
```

[illegible]:

```
int:        !IsClassT
MyClass:   IsClassT
MyStruct: IsClassT
MyUnion:   IsClassT
```

```
enum:       !IsClassT
myfunc(): !IsClassT
```

15.2.3 [illegible]

[illegible]

```
// traits/apply1.hpp
template <typename T>
void apply (T& arg, void (*func)(T))
{
    func(arg);
}
```

[illegible]

```
// traits/apply1.cpp
#include <iostream>
#include "apply1.hpp"
void incr (int& a)
{
    ++a;
}
void print (int a)
{
    std::cout << a << std::endl;
}
int main()
{
    int x=7;
    apply (x, print);
```

```
    apply (x, incr);
}
```

[illegible]

```
apply (x, print)
```

[illegible]int[illegible]T[illegible]apply[illegible]int&[illegible]void(*)(int)[illegible]

```
apply (x, incr)
```

[illegible]2[illegible]int&[illegible]T[illegible]1[illegible]int& &[illegible]int& &[illegible]C++[illegible]C++[illegible] [Standard02][illegible]int&[illegible]T[illegible]T&[illegible]int&[illegible] [9] [illegible]

[illegible]C++[illegible]“[illegible]” [illegible] const [illegible] [10] [illegible]

```
// traits/typeop1.hpp
template <typename T>
class TypeOp {              // [illegible]
  public:
    typedef T           ArgT;
    typedef T           BareT;
    typedef T const     ConstT;
    typedef T &         RefT;
    typedef T &         RefBareT;
    typedef T const & RefConstT;
};
```

□□□□□□□□□□const□□□□□□

```
// traits/typeop2.hpp
template <typename T>
class TypeOp <T const> {   // □□const□□□□□
   public:
      typedef T const    ArgT;
      typedef T             BareT;
typedef T const    ConstT;
typedef T const & RefT;
typedef T &          RefBareT;
typedef T const & RefConstT;
};
```

□□□□□□□□□□□□□ reference-to-const □□□□□□□□□□□□□□□□□□□□Typeop□□□□□□□bare type□□□□□□□□□const□□□□□□□□□C++□□□□□□□□const□□□□□□□□□const□□□□□□□□□□□□□□□□□□const□□□

```
// traits/typeop3.hpp
template <typename T>
class TypeOp <T&> {         // □□□□□□□
   public:
      typedef T &                                ArgT;
      typedef typename TypeOp<T>::BareT   BareT;
      typedef T const                          ConstT;
      typedef T &                                  RefT;
      typedef typename TypeOp<T>::BareT & RefBareT;
      typedef T const &                          RefConstT;
};
```

□□□□□□□□□□□□□□□□□□□□void□□□□□□□□□□□□□□□□□□□□□□□□□void□□□
□□□□□□□□□void□□□□□□□□□□□□□□□□□□□

```
// traits/typeop4.hpp
template<>
class TypeOp <void> {      //□□void□□□□□□
  public:
    typedef void         ArgT;
    typedef void         BareT;
    typedef void const ConstT;
    typedef void         RefT;
    typedef void         RefBareT;
    typedef void         RefConstT;
};
```

□□□□□□□□□□□□□□□□□□□□□apply□□□□□□

```
template <typename T>
void apply (typename TypeOp<T>::RefT arg, void (*func)
(T))
{
  func(arg);
}
```

□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□1□□□□□□□□□T□□□□□T□□□□□□□□□□□□□□
□□□□□□□□□□□□2□□□□□□□□T□□□□□□□□□□□□□□□□□□□□1□□□□□□□□□□□

15.2.4 promotion trait

□□
□□□ promotion

trait [11] [illegible]

[illegible]Array[illegible]

```
template<typename T>
Array<T> operator+ (Array<T> const&, Array<T> const&);
```

[illegible]char[illegible]int[illegible]

```
template<typename T1, typename T2>
Array<???> operator+ (Array<T1> const&, Array<T2> const&);
```

[illegible]promotion trait[illegible]

```
template<typename T1, typename T2>
Array<typename Promotion<T1, T2>::ResultT>
operator+ (Array<T1> const&, Array<T2> const&);
```

[illegible]

```
template<typename T1, typename T2>
typename Promotion<Array<T1>, Array<T2> >::ResultT
operator+ (Array<T1> const&, Array<T2> const&);
```

[illegible] Promotion [illegible]promotion trait[illegible]max()[illegible]“[illegible]”[illegible]2.3[illegible]

[illegible] Promotion [illegible]

```
template<typename T1, typename T2>
class Promotion;
```

□□
□□□□□□IfThenElse□□□□□□□□□□□□Boolean□□□□□□□□□□□□
Boolean□□□□□□□□□□□□□□□□□

```
// traits/ifthenelse.hpp
#ifndef IFTHENELSE_HPP
#define IFTHENELSE_HPP
// □□□□: □□□1□□□□□□□□□2□□□□□3□□
template<bool C, typename Ta, typename Tb>
class IfThenElse;
// □□□□true□□□□□2□□
template<typename Ta, typename Tb>
class IfThenElse<true, Ta, Tb> {
  public:
    typedef Ta ResultT;
};
// □□□□false□□□□□3□□
template<typename Ta, typename Tb>
class IfThenElse<false, Ta, Tb> {
  public:
    typedef Tb ResultT;
};
#endif // IFTHENELSE_HPP
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□T1□T2□void□□□□□
□□□□□□Promotion□□□□

```
// traits/promote1.hpp
// □□□□□□type promotion□□□□□
template<typename T1, typename T2>
```

```
class Promotion {
  public:
    typedef typename
      IfThenElse<(sizeof(T1)>sizeof(T2)),
          T1,
          typename IfThenElse<(sizeof(T1)<sizeof(T2)),
                  T2,
                  void
                >::ResultT
          >::ResultT ResultT;
};
```

[illegible]

```
// traits/promote2.hpp
// [illegible]
template<typename T>
class Promotion<T,T> {
  public:
    typedef T ResultT;
};
```

[illegible]

```
// traits/promote3.hpp
#define MK_PROMOTION(T1,T2,Tr)            \
  template<> class Promotion<T1, T2> {   \
```

```
    public:                                     \
      typedef Tr ResultT;                       \
  };                                            \
                        \
  template<> class Promotion<T2, T1> {   \
    public:                                     \
      typedef Tr ResultT;                       \
  };
```

[illegible]

```
// traits/promote4.hpp
MK_PROMOTION(bool, char, int)
MK_PROMOTION(bool, unsigned char, int)
MK_PROMOTION(bool, signed char, int)
...
```

[illegible] IsFundaT [illegible] IsEnumT [illegible] Promotion [illegible]

[illegible] Promotion [illegible] Array [illegible]

```
// traits/promotearray.hpp
template<typename T1, typename T2>
class Promotion<Array<T1>, Array<T2> > {
  public:
    typedef                                   Array<typename
Promotion<T1,T2>::ResultT> ResultT;
};
```

```
template<typename T>
class Promotion<Array<T>, Array<T> > {
  public:
    typedef Array<typename Promotion<T,T>::ResultT>
ResultT;
};
```

[illegible]Promotion<T,T>[illegible]Promotion<Array<T1>, Array<T2> >[illegible]Promotion<T, T>[illegible]12.4[illegible] [12] [illegible]

[illegible] Promotion [illegible]

15.3 polliicy traiit

[illegible]trait[illegible]trait[illegible]property trait[illegible]

[illegible]trait[illegible]trait[illegible]policy trait[illegible]policy class[illegible]trait[illegible]policy[illegible]policy trait[illegible]policy class[illegible]

[illegible]property trait[illegible]policy trait[illegible]policy[illegible]policy[illegible]

15.3.1 [illegible]

[illegible]C[illegible]C++[illegible]“[illegible]”[illegible]

[illegible]“[illegible]const[illegible]”[illegible]C[illegible]const[illegible]

[illegible]“const[illegible]”[illegible]

[illegible]policy trait[illegible]policy trait[illegible]T[illegible]T[illegible]T const&[illegible]“2 [illegible]”[illegible]“[illegible]”[illegible]“[illegible]const[illegible]”[illegible]

```
template<typename T>
class RParam {
  public:
    typedef typename IfThenElse<sizeof(T)
<=2*sizeof(void*),
                 T,
                 T const&>::ResultT Type;
};
```

[illegible]sizeof[illegible]

```
template<typename T>
class RParam<Array<T> > {
  public:
```

```
        typedef Array<T> const& Type;
    };
```

□□□□□□□□□C++□□□□□□□□□□□□□□□□□□□□□□□class□□□nonclass type□□□□□□□□□□□□□□□□□□□□□□□□class□□□□□□□□□□□□□□“□□”□□□□□□□□□□□15.2.2□□□IsClassT<>□□□□□□□□class□□□□

```
// traits/rparam.hpp
#ifndef RPARAM_HPP
#define RPARAM_HPP
#include "ifthenelse.hpp"
#include "isclasst.hpp"
template<typename T>
class RParam {
  public:
    typedef typename IfThenElse<IsClassT<T>::No,
                       T,
                       T const&>::ResultT Type;
};
#endif // RPARAM_HPP
```

□□□□□□□□□□□□□□□□□□trait□□□□□□□□policy□□□□□□□□□□□policy□□

```
// traits/rparamcls.hpp
#include <iostream>
#include "rparam.hpp"
class MyClass1 {
  public:
```

```
        MyClass1 () {
        }
        MyClass1 (MyClass1 const&) {
            std::cout << "MyClass1 copy constructor called\n";
        }
};
class MyClass2 {
    public:
        MyClass2 () {}
        MyClass2 (MyClass2 const&) {
            std::cout << "MyClass2 copy constructor called\n";
        }
};
// □□RParam<>□MyClass2□□□□□□□□□□□□□□
template<>
class RParam<MyClass2> {
    public:
        typedef MyClass2 Type;
};
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ RParam<>□□□□□□□□□□□□

```
// traits/rparam1.cpp
#include "rparam.hpp"
#include "rparamcls.hpp"
// □□□□□□□□□□□□□□□□□□□□□□
template <typename T1, typename T2>
void foo (typename RParam<T1>::Type p1,
            typename RParam<T2>::Type p2)
```

```
{
      …
}
int main()
{
   MyClass1 mc1;
   MyClass2 mc2;
   foo<MyClass1,MyClass2>(mc1,mc2);
}
```

[illegible] RParam [illegible] foo() [illegible]

[illegible] wrapper [illegible] foo_core() [illegible]

```
// traits/rparam2.cpp
#include "rparam.hpp"
#include "rparamcls.hpp"
// [illegible]
template <typename T1, typename T2>
void foo_core (typename RParam<T1>::Type p1,
               typename RParam<T2>::Type p2)
{
   …
}
// [illegible] wrapper
template <typename T1, typename T2>
```

```
inline
void foo (T1 const & p1, T2 const & p2)
{
  foo_core<T1,T2>(p1,p2);
}
int main()
{
  MyClass1 mc1;
  MyClass2 mc2;
  foo(mc1,mc2);  // [illegible] foo_core<MyClass1,MyClass2>
(mc1,mc2)
}
```

15.3.2 [illegible]

[illegible]policy trait[illegible]

[illegible]

[illegible]

```
T tmp(a);
a = b;
b = tmp;
```

[illegible]20[illegible]

[illegible]trait[illegible]class[illegible]nonclass[illegible]nonclass[illegible]

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□trait□□□□□□□

```
// traits/csmtraits.hpp
template <typename T>
class CSMtraits : public BitOrClassCSM<T, IsClassT<T>::No > {
};
```

CSMtraits□□□□□□□□□□BitOrClassCSM<>□□□□□□CSM□□“copy□swap□move”□□1□□□□□□□□□□□2□□class□□□□□□□□□□□□□□□□□POD□plain old data type□□□□□□□□□□□CSMtraits□□□□□□□□

```
template<>
class CSMtraits<MyPODType>
   : public BitOrClassCSM<MyPODType, true> {
};
```

□□□□□BitOrClassCSM □□□

```
// traits/csm1.hpp
#include <new>
#include <cassert>
#include <stddef.h>
#include "rparam.hpp"
// □□□□template<typename T, bool Bitwise>
class BitOrClassCSM;
// □□□□□□□□□□□□
template<typename T>
class BitOrClassCSM<T, false> {
```

```
    public:
  static void copy (typename RParam<T>::ResultT src, T*
dst) {
    // □□□□□□□□□□□□□□□
    *dst = src;
  }
  static void copy_n (T const* src, T* dst, size_t n) {
    // □□□n□□□□□□□n□
    for (size_tk=0;k<n; ++k) {
      dst[k] = src[k];
    }
  }
  static void copy_init (typename RParam<T>::ResultT src,
                         void* dst) {
    // □□□□□□□□□□□□□□□□
    ::new(dst) T(src);
  }
  static void copy_init_n (T const* src, void* dst, size_t n) {
    // □□n□□□□□□□□□□□□□
    for (size_tk=0;k<n; ++k) {
      ::new((void*)((char*)dst+k)) T(src[k]);
    }
  }
  static void swap (T* a, T* b) {
    // □□□□□□
    T tmp(*a);
    *a = *b;
```

```
    *b = tmp;
}
static void swap_n (T* a, T* b, size_t n) {
    // □□n□
    for (size_tk=0;k<n; ++k) {
        T tmp(a[k]);
        a[k] = b[k];
        b[k] = tmp;
    }
}
static void move (T* src, T* dst) {
    // □□□□□□□□□□□□□
    assert(src != dst);
    *dst = *src;
    src->~T();
    }
    static void move_n (T* src, T* dst, size_t n) {
        // □□n□□□n□□□□□□
        assert(src != dst);
        for (size_tk=0;k<n; ++k) {
            dst[k] = src[k];
            src[k].~T();
        }
    }
    static void move_init (T* src, void* dst) {
        // □□□□□□□□□□□□□□
        assert(src != dst);
```

```
            ::new(dst) T(*src);
            src->~T();
        }
        static void move_init_n (T const* src, void* dst, size_t
  n) {
            // [illegible]n[illegible]
            assert(src != dst);
            for (size_tk=0;k<n; ++k) {
                ::new((void*)((char*)dst+k)) T(src[k]);
                src[k].~T();
            }
        }
    };
```

[illegible]move[illegible]copy [illegible]memcpy()[illegible]memmove()[illegible]memcpy()[illegible]memmove()[illegible]C [illegible] C [illegible]move [illegible]copy[illegible]CSM[illegible]shift[illegible]policy[illegible]memmove()[illegible]shift[illegible]

[illegible]policy trait[illegible]trait class[illegible]

[illegible]trait[illegible]

```
// traits/csm2.hpp
#include <cstring>
#include <cassert>
```

```
#include <stddef.h>
#include "csm1.hpp"
// □□□□□□□□□□□□□□□□□□□□
template <typename T>
class BitOrClassCSM<T,true> : public
BitOrClassCSM<T,false> {
  public:
    static void copy_n (T const* src, T* dst, size_t n) {
      // □□n□□□□□□□
      std::memcpy((void*)dst, (void*)src, n);
    }
    static void copy_init_n (T const* src, void* dst, size_t_t
n) {
      // □□n□□□□□□□□□□□
      std::memcpy(dst, (void*)src, n);
    }
    static void move_n (T* src, T* dst, size_t n) {
      // □□n□□□□□□□n□
      assert(src != dst);
      std::memcpy((void*)dst, (void*)src, n);
    }
    static void move_init_n (T const* src, void* dst, size_t
n) {
      // □□n□□□□□□□□□□□
      assert(src != dst);
      std::memcpy(dst, (void*)src, n);
    }
```

};

[illegible]trait[illegible]

15.4 [illegible]

Nathan Myers[illegible]trait[illegible]trait[illegible]trait[illegible]C++[illegible]trait[illegible]baggage template[illegible]baggage template[illegible]trait[illegible]C++[illegible] baggage [illegible]trait[illegible]trait[illegible]

[illegible]trait[illegible]trait[illegible]trait[illegible]trait[illegible]trait[illegible]typedef[illegible]trait[illegible]

[illegible]trait[illegible] policy trait[illegible]trait[illegible]

```
template <typename T, void (*Policy)(T const&, T const&)>
class X;
```

[illegible]trait[illegible]trait[illegible]Myer[illegible]baggage[illegible]trait[illegible]22[illegible]

[illegible]std::char_traits[illegible]policy trait[illegible]STL[illegible]

std::iterator_traits □□trait□property trait□□□□□□□□□□□□□□□□□□□trait□□□□□□□std::numeric_limits□□□□□□□□□□trait□□□□□□□□□□□□□□□□□□□□□□□□□□□□□std::unary_function□std::binary_funtion□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□typedef□□□□□□□□□□□□□□□□funtor□□□□□□□□□□□22□□□□□□□□□□□□□□□□□□□□□□□□□policy trait□□□□□□std::allocator□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□policy class□□□Alexandrescu□policy class□□□□□□□□□□□□□Modern C++Design□□□□□□□□□□□□policy class □[AlexandrescuDesign]□□

□16□ □□□□□

□□□□□□□□□□□□□□“□□”□□□□□□□□□□□□□□□□□□□□□□□□□□9□□□□□□□□□□dependent base class□□□□□□□□□□unqualified names□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□parameterized inheritance□□□□□□□□□□□□□□□□□□□□□□□□□□□□

16.1 □□□□□□

□□

```
template <typename Policy1 = DefaultPolicy1,
          typename Policy2 = DefaultPolicy2,
          typename Policy3 = DefaultPolicy3,
          typename Policy4 = DefaultPolicy4>
class BreadSlicer {
```

```
    ...
};
```

□□□□□□□□□□□ BreadSlicer<>□□□

□ □ □ □ BreadSlicer<DefaultPolicy1, DefaultPolicy2, Custom>□□□BreadSlicer<Policy3= Custom>□□□□□□□□□□□□□□□□□□□□□ [13] □

□□□BreadSlicer<Policy3_is<Custom> >□□□□□□□□□□□□□□□□□□□□4□policy□□□□□□□□□□□□□□□□□

```
template <typename PolicySetter1 = DefaultPolicyArgs,
        typename PolicySetter2 = DefaultPolicyArgs,
        typename PolicySetter3 = DefaultPolicyArgs,
        typename PolicySetter4 = DefaultPolicyArgs>
class BreadSlicer {
    typedef PolicySelector<PolicySetter1, PolicySetter2,
            PolicySetter3, PolicySetter4>
        Policies;
    // □□Policies::P1□Policies::P2...□□□□policies
    ...
};
```

□□□□□□□□□□ PolicySelector□□□□□□□□□ typedef □□□□□□□□□□□□□□□ Disoriminator□□□□□□□□□□□□□□□policy1-is □ Policy □ □ □ □ □ □ □ □ □ typedef □ □ □ □ Default Policies □ DefaultPolicy1□□□□□□□□□□□□□

```
// PolicySelector<A,B,C,D> □□ A,B,C,D□□□□
```

```
// Discriminator<> □Policy Seletor□□□□□□□□□□□□
template<typename Base, int D>
class Discriminator : public Base {
};
template <typename Setter1, typename Setter2,
      typename Setter3, typename Setter4>
class PolicySelector : public Discriminator<Setter1,1>,
            public Discriminator<Setter2,2>,
            public Discriminator<Setter3,3>,
            public Discriminator<Setter4,4> {
};
```

□□□□□□□□□Discriminator□□□□□□□□□□□□□□□Setter□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ [14] □

□□□□□□□□□□□□□□□□□□□□□□

```
// □□□□□□policies□ P1, P2, P3, P4
class DefaultPolicies {
  public:
    typedef DefaultPolicy1 P1;
    typedef DefaultPolicy2 P2;
    typedef DefaultPolicy3 P3;
    typedef DefaultPolicy4 P4;
};
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
// □□□□□□□□policy□□□
// □□□□□□□□□DefaultPolicies□□□□□□□□□□□□□□□
class DefaultPolicyArgs : virtual public DefaultPolicies {
};
```

□□□□□□□□□□□□□□□□□□policy□□□

```
template <typename Policy>
class Policy1_is : virtual public DefaultPolicies {
   public:
      typedef Policy P1; // □□□□□ typedef
};
template <typename Policy>
class Policy2_is : virtual public DefaultPolicies {
   public:
      typedef Policy P2; //□□□□□ typedef
};
template <typename Policy>
class Policy3_is : virtual public DefaultPolicies {
   public:
      typedef Policy P3; //□□□□□ typedef
};
template <typename Policy>
class Policy4_is : virtual public DefaultPolicies {
   public:
      typedef Policy P4; //□□□□□ typedef
};
```

□□□□□□□□□□BreadSlicer□□□□□

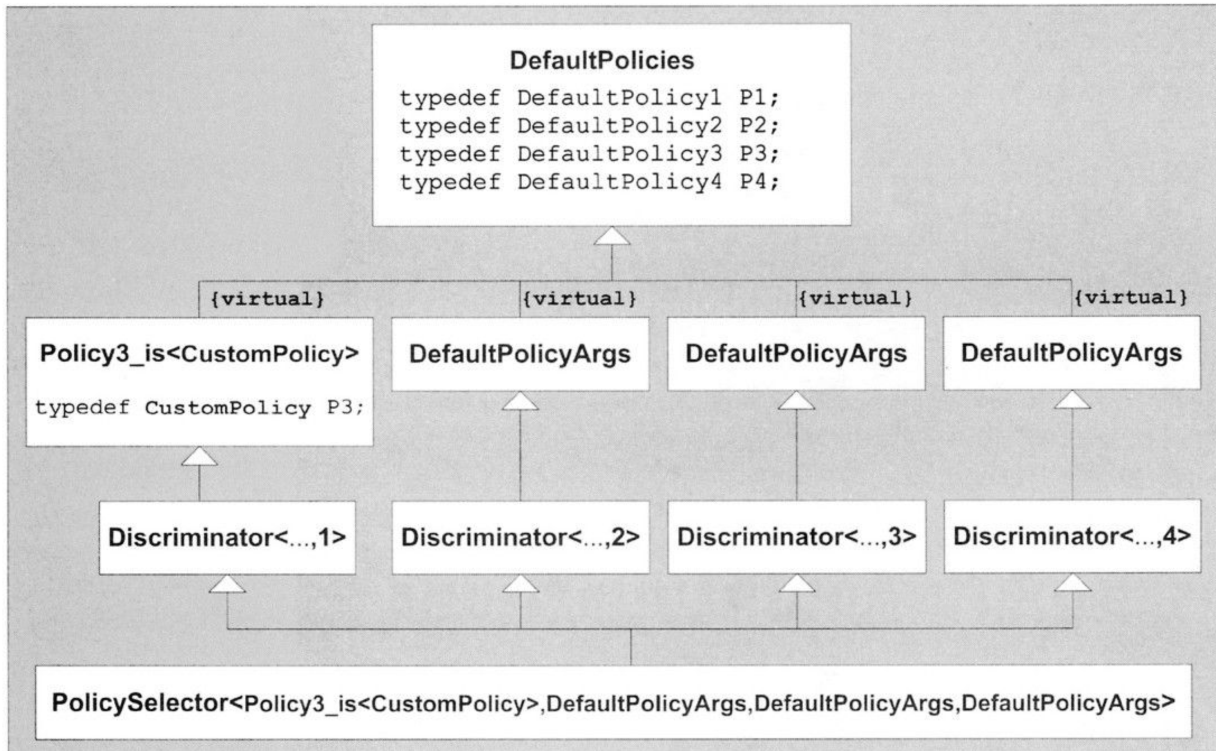
```
BreadSlicer<Policy3_is<CustomPolicy> > bc;
```

□□□□BreadSlicer□□□□Polices□□□□□

```
PolicySelector<Policy3_is<CustomPolicy>,
            DefaultPolicyArgs,
            DefaultPolicyArgs,
```

DefaultPolicyArgs>

□□□□Discriminator□□□□□□□□□□□16.1□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□DefaultPolicies□□□□□□□□□□□□□□P1□P2□P3□P4□□□□□□□□□□□□□□□□□□□Policy3_is<>□□□□P3□□□□□□□□□domination rule□□□□□□□□□□□□□□□□□□□□□□□□□□[15]□



□16.1 BreadSlicer<>::Policies□□□□□□□

□□□BreadSlicer□□□□□□□□□□Policies::P3□□□□□□qualified name□□□□□4□policy□□□□

```
template <...>
class BreadSlicer {
  ...
public:
  void print () {
    Policies::P3::doPrint();
```

```
    }
    ...
};
```

□□□□□□inherit/nametmpl.cpp□

□□□□□□□□□□4□□

16.2 □□□□□

C++□□□□“□”□□□

□□□□□□□□□□□□□0□□□□□□□□□□

```
// inherit/empty.cpp
#include <iostream>
class EmptyClass {
};
int main()
{
    std::cout << "sizeof(EmptyClass): " << sizeof(EmptyClass)
              << '\n';
}
```

□□□□□□□□□□□□□□EmptyClass□□□□1□□□□□□□□□□alignment□□□□□□□□□□□□□□□□□□□□□□□4□□

16.2.1 □□□□

C++□□□□□□□□□□□□□ 0□□□□□□□□□□□□□□□□□□□□□□□□□0□□□□□□□□□□□□□□□□□□□□□□□□□□ZeroSizedT□□□□0□□□□□□□□□□□□□

```
ZeroSizedT z[10];
...
&z[i] - &z[j]       // [illegible]
```

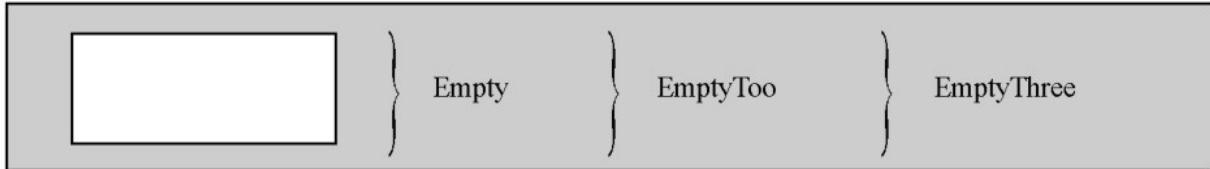
[illegible]0[illegible]

[illegible]“[illegible]”[illegible]C++[illegible]empty base class optimization[illegible]EBCO[illegible]

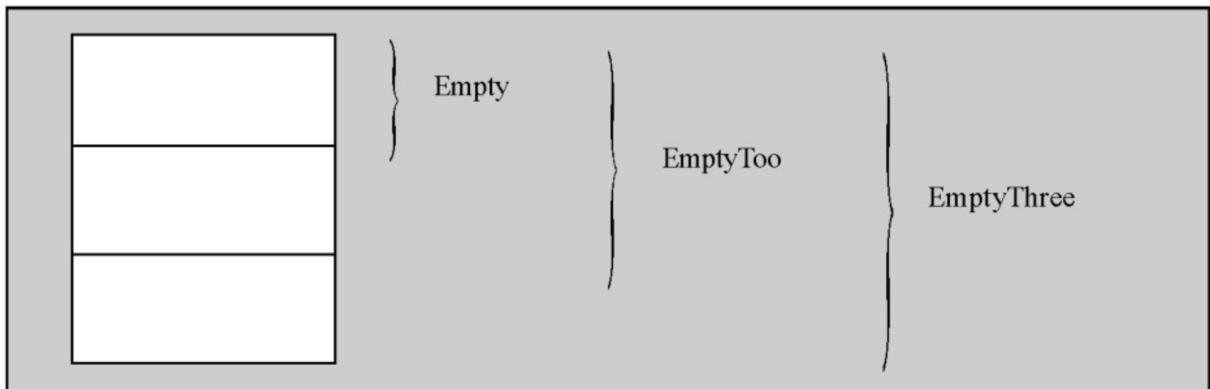
```
// inherit/ebco1.cpp
#include <iostream>
class Empty {
    typedef int Int; // typedef[illegible]
};
class EmptyToo : public Empty {
};
class EmptyThree : public EmptyToo {
};
int main()
{
    std::cout << "sizeof(Empty): " << sizeof(Empty)
      << '\n';
    std::cout << "sizeof(EmptyToo): " << sizeof(EmptyToo)
      << '\n';
    std::cout << "sizeof(EmptyThree): " << sizeof(EmptyThree)
      << '\n';
```

```
}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□0□□□16.2□□□□□□□□□EmpytToo□□□Empty□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□0□□□□□EmptyThree□□□□Empty□□□□□□□□□□□□□□□□□□□EBCO□□□□□□□□□□□□□□□16.3□□



□16.2 □□EBCO□□□□□EmptyThree□□□



□16.3 □□□EBCO□□□□□EmptyThree□□□

□□□□□□□□□□□□□□□□□□

```
// inherit/ebco2.cpp
#include <iostream>
class Empty {
   typedef int Int; // typedef□□□□□□□□□□□□□
};
class EmptyToo : public Empty {
};
class NonEmpty : public Empty, public EmptyToo {
};
```

```
int main()
{
    std::cout << "sizeof(Empty): " << sizeof(Empty) << '\n';
    std::cout << "sizeof(EmptyToo): " << sizeof(EmptyToo) << '\n';
    std::cout << "sizeof(NonEmpty): " << sizeof(NonEmpty) << '\n';
}
```

□□□□□□□□NonEmpty□□□□“□”□□□□□□□□□□□□□□□□□□□NonEmpty□□□Empty□ EmptyToo□□□□□□□□□□□□□□EmptyToo□□□Empty□□NonEmpty□□□Empty□□□□□□□□□□□□□□□□□□□□□□□□□□C++□□□□□□□□□□□□□□□□□Empty□□□□□□□□0□1□□□□□□□□□□□□□1□□□□□□□□NonEmpty□□□□□□□□□□□□□□□Empty□□□□□□□□□□□□□16.4□□

□16.4 □□EBCO□□□□NonEmpty□□□

□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□typedefs□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

16.2.2 □□□□□

[illegible]

[illegible]

```
template <typename T1, typename T2>
class MyClass {
  private:
    T1 a;
    T2 b;…
};
```

[illegible]T1[illegible]T2[illegible]MyClass<T1, T2>[illegible]

[illegible]

```
template <typename T1, typename T2>
class MyClass : private T1, private T2 {
};
```

[illegible]T1[illegible]T2[illegible]int[illegible]union[illegible]T1[illegible]T2[illegible] [16] [illegible]MyClass[illegible]EBCO[illegible]

[illegible]EBCO[illegible]“[illegible]”[illegible] [17] [illegible]

```
template <typename CustomClass>
class Optimizable {
```

```
    private:
        CustomClass info;      // □□□□
        void*          storage;
        ...
};
```

□□□□□□□

```
template <typename CustomClass>
class Optimizable {
    private:
        BaseMemberPair<CustomClass, void*> info_and_storage;
        ...
};
```

□□□□□BaseMemberPair□□□□□Optimizable□□□

BaseMemberPair□□□□□□□□

```
// inherit/basememberpair.hpp
#ifndef BASE_MEMBER_PAIR_HPP
#define BASE_MEMBER_PAIR_HPP
template <typename Base, typename Member>
class BaseMemberPair : private Base {
    private:
        Member member;
    public:
        // □□□□
        BaseMemberPair (Base const & b, Member const & m)
```

```
        : Base(b), member(m) {
    }
    // □□first()□□□□□□□
    Base const& first() const {
        return (Base const&)*this;
    }
    Base& first() {
        return (Base&)*this;
    }
    // □□second()□□□□□□□□□□
    Member const& second() const {
        return this->member;
    }
    Member& second() {
        return this->member;
    }
};
#endif // BASE_MEMBER_PAIR_HPP
```

□□□BaseMemberPair□□□□□□□□□□□□□□□Base□□□□□□□□□□□□□□□□□□first()□second()□□□

16.3 □□□□□□□□□

□□□□□□□□□□Curiously Recurring Template Pattern□CRTP□□

```
template <typename Derived>
```

```
class CuriousBase {
  ...
};
class Curious : public CuriousBase<Curious> {
  ...
};
```

[illegible]CRTP[illegible]nondependent base class[illegible]Curious[illegible]dependent base class[illegible]CRTP[illegible]

```
template <typename Derived>
class CuriousBase {
  ...
};
template <typename T>
class CuriousTemplate : public CuriousBase<CuriousTemplate<T> > {
  ...
};
```

[illegible]

```
template <template<typename> class Derived>
class MoreCuriousBase {
  ...
};
template <typename T>
class MoreCurious : public MoreCuriousBase<MoreCurious> {
  ...
```

};

CRTP [illegible]
CRTP[illegible]

```
// inherit/objectcounter.hpp
#include <stddef.h>
template <typename CountedType>
class ObjectCounter {
  private:
    static size_t count;    // [illegible]
  protected:
    // [illegible]
    ObjectCounter() {
      ++ObjectCounter<CountedType>::count;
}
  // [illegible]
  ObjectCounter (ObjectCounter<CountedType> const&) {
    ++ObjectCounter<CountedType>::count;
  }
  // [illegible]
  ~ObjectCounter() {
    --ObjectCounter<CountedType>::count;
  }
public:
  // [illegible]:
  static size_t live() {
```

```
        return ObjectCounter<CountedType>::count;
    }
};
// □0□□□count
template <typename CountedType>
size_t ObjectCounter<CountedType>::count = 0;
```

□□□□□□□□□□□□□□□□□□□□ObjectCounter□□□□□□□□□
□□□

```
// inherit/testcounter.cpp
#include "objectcounter.hpp"
#include <iostream>
template <typename CharT>
class MyString : public ObjectCounter<MyString<CharT>
> {
    ...
};
int main()
{
    MyString<char> s1, s2;
    MyString<wchar_t> ws;
    std::cout << "number of MyString<char>: "
            << MyString<char>::live() << std::endl;
    std::cout << "number of MyString<wchar_t>: "
            << ws.live() << std::endl;
}
```

□□□CRTP □□□□□□□□□□□□□□□□□□□□□□□□operator[]
□□□□□□□□

16.4 [illegible]

C++[illegible]3[illegible]nontype[illegible]

[illegible]

```
// inherit/virtual.cpp
#include <iostream>
class NotVirtual {
};
class Virtual {
  public:
    virtual void foo() {
    }
};
template <typename VBase>
class Base : private VBase {
  public:
    // foo()[illegible]VBase([illegible])[illegible]
    void foo() {
      std::cout << "Base::foo()" << '\n';
    }
};
template <typename V>
class Derived : public Base<V> {
  public:
    void foo() {
      std::cout << "Derived::foo()" << '\n';
    }
```

```
};
int main()
{
  Base<NotVirtual>* p1 = new Derived<NotVirtual>;
  p1->foo(); // □□Base::foo()
  Base<Virtual>* p2 = new Derived<Virtual>;
  p2->foo(); // □□ Derived::foo()
}
```

□□ [18] □

16.5 □□□□

Boost□□□□□□□□□□□□□ [19] □□□□□□□□□□□□□□□□ PolicySelector □□□□ Vandevoorde □□□□□Boost □□□□□□metaprogramming□□□

CRTP□□□□□□□1991□□□□□James Coplien□□□□□□□□[CoplienCRTP] □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ parameterized inheritance□□□□□□□□□CRTP□□□□□□□□□□CRTP□□□□□□□□□□□□□□□□□□□CRTP□□□CRTP□□□Barton-Nackman□□□□11.7□□□□□□□□Barton□Nackman□□□□□□□□friend name injection□Barton-Nackman□□□□□□□□□CRTP□□□□ObjectCounter □ □ □ □ □ □ □ □ □ □ Scott Meyers □ □ □ □ □ □ [MeyersCounting]□□□

Bill Gibbons□□EBCO□□□C++□□□□Nathan Myers□EBCO□□□□□□□□□□□ BaseMemberPair □□□□Boost □□□□□□□

□□compressed_pair□□□□□□□□□□□□□□□□□MyClass□□□□□□boost□compressed-pair□□□□□□BaseMemberPair□

□□□□

□□□□□□

□□□□□□□□□□□□□□□named function parameter□□□□□□□□□□□□□Python□□□□□□□□□□□keyword argument□□C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□Python□□□□

def parrot(voltage, state='a stiff', action='voom', type='Norwegian Blue'):

print "-- This parrot wouldn't", action,

print "if you put", voltage, "Volts through it."

print "-- Lovely plumage, the", type

print "-- It's", state, "!"

□□□□□

parrot(1000)

parrot(action = 'VOOOOOM', voltage = 1000000)

parrot('a thousand', state = 'pushing up the daisies')

parrot('a million', 'bereft of life', 'jump')

□□□□□□□□□□□□□□□□□□□placeholder□□□C++□□□C++□□□□□□□□□□□□□BGL□Boost Graph Library□□□□□□□□□□□□□□□□□□Bellman-Ford□□□□□□□□□□□□□□

template <class EdgeListGraph, class Size, class WeightMap,

class PredecessorMap, class DistanceMap,

```
class BinaryFunction, class BinaryPredicate,
class BellmanFordVisitor>
bool bellman_ford_shortest_paths(EdgeListGraph& g, Size N,
WeightMap weight, PredecessorMap pred, DistanceMap distance,
BinaryFunction combine, BinaryPredicate compare, BellmanFordVisitor v);
```

8 6

```
bool r = boost::bellman_ford_shortest_paths(g, int(N),
boost::weight_map(weight).
distance_map(&distance[0]).
predecessor_map(&parent[0]));
```

BGL Boost Graph Library

17 metaprogram

metaprogramming [20] “ ”

metaprogramming metaprogramming

metaprogramming

metaprogramming metaprogramming

[illegible]metaprogram[illegible]

metaprogramming[illegible]15[illegible]trait[illegible]15[illegible]

17.1 metaprogram[illegible]

[illegible]1994[illegible]C++[illegible]Erwin Unruh[illegible]2[illegible]template metaprogramming[illegible]

[illegible]metaprogramming[illegible]Erwin[illegible]17.8[illegible]3[illegible]

```
// meta/pow3.hpp
#ifndef POW3_HPP
#define POW3_HPP
// [illegible]3[illegible]N[illegible]
template<int N>
class Pow3 {
  public:
    enum { result=3*Pow3<N-1>::result };
};
// [illegible]
template<>
class Pow3<0> {
```

```
   public:
      enum { result = 1 };
};
#endif // POW3_HPP
```

□□□□□template metaprogramming□□□□□□□□□□□□□□□□

[21] □□□□□3^N □□□□□□□□□□□□□□□□

1. $3^N = 3 * 3^{N-1}$

2. $3^0 = 1$

□□□□1□□□□□□□□□□□□□□

```
template<int N>
class Pow3 {
   public:
      enum { result = 3 * Pow3<N-1>::result };
};
```

□□□□□□□N□□□□□Pow3<>□□□□□□□□□□□□□□□□□□N-1□□□□□□□□□□□□□□□□result□□3□

□□2□□□□□□□□□□□□□□□□□Pow3<0>□□□

```
template<>
class Pow3<0> {
   public:
      enum { result = 1 };
};
```

□□□□□□□Pow3<7>□□□3^7 □□□□□□□□□□□□□

```
// meta/pow3.cpp
#include <iostream>
#include "pow3.hpp"
```

```
int main()
{
    std::cout << "Pow3<7>::result = " << Pow3<7>::result
              << '\n';
}
```

[illegible]Pow3<7>[illegible]

3 * Pow3<6>::result

[illegible]Pow3<6>[illegible]6[illegible]Pow3<6>[illegible]Pow3<5>[illegible]Pow3<4>[illegible]Pow3<>[illegible]0[illegible]1[illegible]Pow3<0>[illegible]

[illegible] Pow3<> [illegible] template metaprogramming[illegible]template metaprogramming[illegible]

17.2 [illegible]

[illegible] C++[illegible]“[illegible]”[illegible]C++[illegible]

```
struct TrueConstants {
    enum { Three = 3 };
    static int const Four = 4;
};
```

[illegible]Four[illegible]“[illegible]”——[illegible]Three[illegible]

[illegible]Pow3 metaprogram[illegible]

```
// meta/pow3b.hpp
#ifndef POW3_HPP
#define POW3_HPP
// □□□□3□N□□□□□□□
template<int N>
class Pow3 {
  public:
    static int const result = 3 * Pow3<N-1>::result;
};
// □□□□□□□□□□□□□
template<>
class Pow3<0> {
  public:
    static int const result = 1;
};
#endif // POW3_HPP
```

□□

```
void foo(int const&);
```

□□□□□□□metaprogram□□□□□□□□□□□

```
foo(Pow3<7>::result);
```

□□□□□□□□□□□Pow3<7>::result□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□“□□□”□□□

□□

17.3 □2□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□N□□□□□metaprogram□□□□□□□□□□□□□□□□□□

```
// meta/sqrt1.hpp
#ifndef SQRT_HPP
#define SQRT_HPP
// □□□□ sqrt(N)□□□□□
template <int N, int LO=0, int HI=N>
class Sqrt {
  public:
    // □□□□
    enum { mid = (LO+HI+1)/2 };
    // □□□□□□□□□□□result enum { result = (N<mid*mid) ?
Sqrt<N,LO,mid-1>::result
                  : Sqrt<N,mid,HI>::result };
};
// □□□□□□□□LO □□ HI
template<int N, int M>
class Sqrt<N,M,M> {
  public:
    enum { result=M};
};
#endif // SQRT_HPP
```

□1□□□□□□□□□□□□□□□□□□□□N□□N□□□□□□□□□□□□□□□□□□□□□□□□□□0□N□□□

[illegible]result[illegible]LO[illegible]HI[illegible] ?: [illegible]mid^2 [illegible]N[illegible]

[illegible]LO[illegible]HI[illegible]M[illegible]M[illegible]

[illegible]metaprogram[illegible]

```
// meta/sqrt1.cpp
#include <iostream>
#include "sqrt1.hpp"
int main()
{
    std::cout << "Sqrt<16>::result = " <<
Sqrt<16>::result
          << '\n';
    std::cout << "Sqrt<25>::result = " <<
Sqrt<25>::result
          << '\n';
    std::cout << "Sqrt<42>::result = "
<<Sqrt<42>::result
          << '\n';
    std::cout << "Sqrt<1>::result = " <<
Sqrt<1>::result
          << '\n';
}
```

[illegible]

Sqrt<16>::result

[illegible]

```
Sqrt<16,1,16>::result
```

□□□□□□□□metaprogram□□Sqrt<16,1,16>::result□□□□□□

```
mid = (1+16+1)/2
   = 9
result = (16<9*9) ? Sqrt<16,1,8>::result
         : Sqrt<16,9,16>::result
      = (16<81) ? Sqrt<16,1,8>::result
      : Sqrt<16,9,16>::result
      = Sqrt<16,1,8>::result
```

□□□□□□□□□□□□Sqrt<16,1,8>::result□□□□□□□

```
mid = (1+8+1)/2
   = 5
result = (16<5*5) ? Sqrt<16,1,4>::result
         : Sqrt<16,5,8>::result
      = (16<25) ? Sqrt<16,1,4>::result
         : Sqrt<16,5,8>::result
      = Sqrt<16,1,4>::result
```

□□□Sqrt<16,1,4>::result□□□□□□□□

```
mid = (1+4+1)/2
   = 3
result = (16<3*3) ? Sqrt<16,1,2>::result
         : Sqrt<16,3,4>::result
      = (16<9) ? Sqrt<16,1,2>::result
         : Sqrt<16,3,4>::result
      = Sqrt<16,3,4>::result
```

□□□Sqrt<16,3,4>::result□□□□□□

```
mid = (3+4+1)/2
```

```
   = 4
result = (16<4*4) ? Sqrt<16,3,3>::result
           : Sqrt<16,4,4>::result
       = (16<16) ? Sqrt<16,3,3>::result
           : Sqrt<16,4,4>::result
       = Sqrt<16,4,4>::result
```

[illegible]Sqrt<16,4,4>::result[illegible]

```
result = 4
```

[illegible]

[illegible] 16 [illegible]

```
(16<=8*8) ? Sqrt<16,1,8>::result
       : Sqrt<16,9,16>::result
```

[illegible]Sqrt<16,1,8>[illegible]Sqrt<16,9,16>[illegible] :: [illegible] result[illegible]Sqrt<16,9,16>[illegible] Sqrt<16,9,12>[illegible] Sqrt<16,13,16>[illegible]N[illegible]

[illegible] ?: [illegible]Sqrt metaprogram[illegible]

```
// meta/sqrt2.hpp
#include "ifthenelse.hpp"
// [illegible]
template<int N, int LO=0, int HI=N>
```

```
class Sqrt {
  public:
    // [illegible]
    enum { mid = (LO+HI+1)/2 };
    // [illegible]
    typedef typename IfThenElse<(N<mid*mid),
              Sqrt<N,LO,mid-1>,
              Sqrt<N,mid,HI> >::ResultT
      SubT;
    enum { result = SubT::result };
};
// [illegible]
template<int N, int S>
class Sqrt<N, S, S> {
  public:
    enum { result = S };
};
```

[illegible]IfThenElse[illegible]15.2.4[illegible]

```
// meta/ifthenelse.hpp
#ifndef IFTHENELSE_HPP
#define IFTHENELSE_HPP
// [illegible]: [illegible]1[illegible]2[illegible]3[illegible]
template<bool C, typename Ta, typename Tb>
class IfThenElse;
// [illegible]: true [illegible]2[illegible]
template<typename Ta, typename Tb>
```

```
class IfThenElse<true, Ta, Tb> {
  public:
    typedef Ta ResultT;
};
// [illegible]: false[illegible]3[illegible]
template<typename Ta, typename Tb>
class IfThenElse<false, Ta, Tb> {
  public:
    typedef Tb ResultT;
};
#endif // IFTHENELSE_HPP
```

[illegible]IfThenElse[illegible]1[illegible]typedef[illegible]ResultT[illegible]ResultT[illegible]2[illegible]typedef[illegible]C++[illegible]

```
typedef typename IfThenElse<(N<mid*mid),
                            Sqrt<N,LO,mid-1>,
                            Sqrt<N,mid,HI> >::ResultT
        SubT;
```

[illegible]Sqrt<N,LO,mid-1>[illegible]Sqrt<N,mid,HI>[illegible]SubT[illegible]Sub::result[illegible]SubT[illegible]$\log_2(N)$[illegible]N[illegible]metaprogramming[illegible]

17.4 [illegible]

[illegible]metaprogram[illegible]
[illegible]metaprogram[illegible]metaprogramming[illegible]
metaprogram[illegible]

[illegible]“[illegible]”[illegible]N[illegible]I[illegible]0[illegible]N[illegible]I[illegible]N[illegible]I[illegible]N[illegible] C++[illegible]

```
int I;
for (I=0; I*I<N; ++I) {
    ;
}
// I[illegible]N[illegible]
```

[illegible] metaprogram[illegible]metaprogram[illegible]

```
meta/sqrt3.hpp
#ifndef SQRT_HPP
#define SQRT_HPP
// [illegible] sqrt(N)[illegible]
template <int N, int I=0>
class Sqrt {
    public:
        enum { result = (I*I<N) ? Sqrt<N,I+1>::result
                          : I };
};
// [illegible]
template<int N>
class Sqrt<N,N> {
```

```
    public:
        enum { result = N };
};
#endif // SQRT_HPP
```

□□□□□ I □□□□□□□□□ I*I<N □□□□□□□□□□□□□□ Sqrt<N,I+1>::result□□□□□□□result□□□□□□□□□I□□□result□□□□

□□□□□□□□Sqrt<16>□□□□□□□□□□□□□Sqrt<16,1>□□□□□□□1□□□□□□□□□□□I□□□□□□□□□□□□□□□□□□□□□□I^2 □□□□I*I□□□N□□□□□□□□□□□ Sqrt<N,I+1>::result□□□□□□□□□□□□□□I^2 □□□□□□N□□□□□□I□□□□□□□□□

□□□ 1 □□□□□□□□□□□□ I□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ ?: □□Sqrt<4>□□□□□□□□□□□□□□□

- □□1:

```
result = (1*1<4) ? Sqrt<4,2>::result
                 : 1
```

- □□2:

```
result = (1*1<4) ? (2*2<4) ? Sqrt<4,3>::result
                           : 2
                 : 1
```

- □□3:

```
result = (1*1<4) ? (2*2<4) ? (3*3<4) ? Sqrt<4,4>::result
                                     : 3
                           : 2
                 : 1
```

• [illegible]4:

```
result = (1*1<4) ? (2*2<4) ? (3*3<4) ? 4
                                    : 3
                               : 2
                          : 1
```

[illegible]Step 2[illegible]

[illegible]IfThenElse[illegible]

```
// meta/sqrt4.hpp
#ifndef SQRT_HPP
#define SQRT_HPP
#include "ifthenelse.hpp"
// [illegible] result [illegible]
template<int N>
class Value {
  public:
    enum { result = N };
};
// [illegible] sqrt(N)[illegible]
template <int N, int I=0>
class Sqrt {
  public:
    //[illegible]Sqrt<N,I+1>[illegible]Value<I>[illegible]
    typedef typename IfThenElse<(I*I<N),
                        Sqrt<N,I+1>,
                        Value<I>
```

```
                    >::ResultT
            SubT;
        // [illegible]
        enum { result = SubT::result };
};
#endif // SQRT_HPP
```

[illegible] Value<>[illegible]result[illegible]

[illegible]IfThenElse<>[illegible]Sqrt(N)[illegible]N[illegible]metaprogramming[illegible]64[illegible]4 096[illegible]64[illegible]

[illegible]Sqrt[illegible]

```
Sqrt<16>::result = 4
Sqrt<25>::result = 5
Sqrt<42>::result = 7
Sqrt<1>::result = 1
```

[illegible]42[illegible]7[illegible]6[illegible]

17.5 [illegible]

Pow3<>[illegible]Sqrt[illegible]template metaprogram[illegible]

- [illegible]
- [illegible]
- [illegible]

• [illegible]

[illegible] metaprogram [illegible] C++ [illegible] 17 [illegible] template metaprogramming [illegible] 17 [illegible]

[illegible] templat metaprogram [illegible] metaprogram [illegible] metaprogram [illegible]

17.6 [illegible]

[illegible]

```
template<typename T, typename U>
struct Doublify {};
template<int N>
struct Trouble {
    typedef Doublify<typename Trouble<N-1>::LongType,
                     typename Trouble<N-1>::LongType> LongType;
};
template<>
struct Trouble<0> {
    typedef double LongType;
};
```

Trouble<10>::LongType ouch;

Trouble<10>::LongType □□□□□□□□ Trouble<9>□ Trouble<8>…Trouble<0>□□□□□□□□□□□□□□□□□□□□□□□□□ Doublify□□□□□□17.1□□□□□□□□

□17.1 Trouble<N>::LongType

TypedefName	Underlying Type
Trouble<0>::LongType	double
Trouble<1>::LongType	Doublify<double,double>
Trouble<2>::LongType	Doublify<Doublify<double,double>, Doublify<double,double>>
Trouble<3>::LongType	Doublify<Doublify<Doublify<double,double>, Doublify<double,double>>, <Doublify<double,double>, Doublify<double,double>>>

□□17.1□□□□□□□□□Trouble<N>::LongType□□□□□□□□□□2□N□□C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□mangled name□□□□mangled name□□□□□□□□□□□□□□□□□□□□□□□□□C++□□□□□mangled name □□□□□□□ template-id □□□□□□□□Trouble<10>::LongType□□□□□□□□□□□□□□□10 000□□□□mangled name□

□□□□□□□□C++□□□□□□□□□□□template-id□□C++□□□□□□□□□template-id□□□□□□□□□□□□□□□□mangled name□□□□□□□□□□□□□□Trouble<10>::LongType□□

17.7 □□metaprogram□□□□□

□□□□□□□□metaprogramming□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□n□□□□□□□□vector□□□□□□□□□□□□□□□□□vector a□b□□□□a□b□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□vector□□3□□□□□□□□□□

a[0]*b[0] + a[1]*b[1] + a[2]*b[2]

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
// meta/loop1.hpp
#ifndef LOOP1_HPP
#define LOOP1_HPP
template <typename T>
inline T dot_product (int dim, T* a, T* b)
{
   T result = T□□;
   for (int i=0; i<dim; ++i) {
      result += a[i]*b[i];
   }
   return result;
}
#endif // LOOP1_HPP
```

□□□□□□□□□□□

```
// meta/loop1.cpp
#include <iostream>
#include "loop1.hpp"
int main()
{
   int a[3] = { 1, 2, 3 };
   int b[3] = { 5, 6, 7 };std::cout << "dot_product(3,a,b)
 = " << dot_product(3,a,b)
```

```
            << '\n';
        std::cout << "dot_product(3,a,a) = " <<
  dot_product(3,a,a)
            << '\n';
    }
```

[illegible]

dot_product(3,a,b) = 38

dot_product(3,a,a) = 14

[illegible]

[illegible]

a[0]*b[0] + a[1]*b[1] + a[2]*b[2]

[illegible]

[illegible]

[illegible] dot_product() [illegible] template metaprogramming [illegible]“[illegible]”[illegible] metaprogam [illegible]

```
// meta/loop2.hpp
#ifndef LOOP2_HPP
#define LOOP2_HPP
// [illegible]
template <int DIM, typename T>
class DotProduct {
  public:
```

```
        static T result (T* a, T* b) {
            return *a * *b + DotProduct<DIM-1,T>::result(a+1,b+1);
        }
};
// [illegible]
template <typename T>
class DotProduct<1,T> {
    public:
        static T result (T* a, T* b) {
            return *a * *b;
        }
};
// [illegible]
template <int DIM, typename T>
inline T dot_product (T* a, T* b)
{
    return DotProduct<DIM,T>::result(a,b);
}
#endif // LOOP2_HPP
```

[illegible]

```
// meta/loop2.cpp
#include <iostream>
#include "loop2.hpp"
int main()
{
    int a[3] = { 1, 2, 3};
```

```
    int b[3] = { 5, 6, 7};
    std::cout << "dot_product<3>(a,b) = " <<
  dot_product<3>(a,b)
      << '\n';
    std::cout << "dot_product<3>(a,a) = " <<
  dot_product<3>(a,a)
      << '\n';
}
```

□□□□□□□□□

```
dot_product(3,a,b)
```

□□□□

```
dot_product<3>(a,b)
```

□□□□□□□□dot_product<3>(a,b) □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
DotProduct<3,int>::result(a,b)
```

□□□□□□□□□□□□□□metaprogram□□□□

□□□metaprogram□□□result□□□1□□□a□b□□□□□□□□vector□□□□□□□□□vector□□□□a□b□□□□□□□□□□□□□a□b□□□□□□vector□□□□

```
template <int DIM, typename T>
class DotProduct {
public:
    static T result (T* a, T* b) {
        return *a * *b + DotProduct<DIM-
1,T>::result(a+1,b+1);
    }
};
```

□□□□□□□□□□vector□□□□

```
template <typename T>
class DotProduct<1,T> {
  public:
    static T result (T* a, T* b) {
      return *a * *b;
    }
};
```

[illegible]

```
dot_product<3>(a,b)
```

[illegible]

```
DotProduct<3,int>::result(a,b)
= *a * *b + DotProduct<2,int>::result(a+1,b+1)
= *a * *b + *(a+1) * *(b+1) + DotProduct<1,int>::result(a+2,b+2)
= *a * *b + *(a+1) * *(b+1) + *(a+2) * *(b+2)
```

[illegible] metaprogram [illegible]vector [illegible]

[illegible]Blitz++ ([illegible] [Blitz++])[illegible]the MTL library ([illegible] [MTL])[illegible]POOMA ([illegible] [POOMA])[illegible] metaprogram[illegible]metaprogram[illegible]metaprogram[illegible] [22] [illegible]

17.8 [illegible]

[illegible]metaprogram[illegible]Erwin Unruh[illegible] C++[illegible]Erwin Unruh [illegible]metaprogram[illegible]Metaware[illegible]1994[illegible]C++[illegible] [23] [illegible]

```
// meta/unruh.cpp
// Erwin Unruh [illegible]
template <int p, int i>
class is_prime {
  public:
    enum { prim = (p==2) || (p%i) && is_prime<(i>2?p:0),i-1>::prim
      };
};
template<>
class is_prime<0,0> {
  public:
    enum {prim=1};
};
template<>
class is_prime<0,1> {
  public:
    enum {prim=1};
};
template <int i>
class D {
  public:
```

```
        D(void*);
};
template <int i>
class Prime_print {       // [illegible]
    public:
        Prime_print<i-1> a;
        enum { prim = is_prime<i,i-1>::prim
            };
        void f() {
            D<i> d = prim ? 1 : 0;
        a.f();
    }
};
template<>
class Prime_print<1> {   // [illegible]
    public:
        enum {prim=0};
        void f() {
            D<1> d = prim ? 1 : 0;
        };
};
#ifndef LAST
#define LAST 18
#endif
int main()
{
    Prime_print<LAST> a;
```

```
    a.f();
}
```

□□□□□□□□□□□□□□Primer_print::f()□□□□□□□□d□□□□□□□□□□□□□□□□□□□□□□□□□□□1□□□□□□□□□□D□□□□□□□□□void*□□□□□□□□□1□□□□□□d□□□□□0□□□□void*□□□□□□□□0□□□□□d□□D<N>□□

unruh.cpp:36: conversion from 'int' to non-scalar type 'D<17>' requested

unruh.cpp:36: conversion from 'int' to non-scalar type 'D<13>' requested

unruh.cpp:36: conversion from 'int' to non-scalar type 'D<11>' requested

unruh.cpp:36: conversion from 'int' to non-scalar type 'D<7>' requested

unruh.cpp:36: conversion from 'int' to non-scalar type 'D<5>' requested

unruh.cpp:36: conversion from 'int' to non-scalar type 'D<3>' requested

unruh.cpp:36: conversion from 'int' to non-scalar type 'D<2>' requested

□□C++ template metaprogramming□□□□Todd Veldhuizen□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ Using C++ Template Metaprograms□□[VeldhuizenMeta95]□□□□□□□□□□□□□□Blitz++□□□□□ C++□□□□□□□□□□[Blitz++]□□□□□□□□metaprogramming □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

[illegible]18[illegible] [illegible]

[illegible]expression template[illegible]

[illegible]scalar[illegible]1[illegible]scalar[illegible]

```
Array<double> x(1000), y(1000);
...
x = 1.2*x + x*y;
```

[illegible]

[illegible]template metaprogramming[illegible] template metaprogramming[illegible]17.7[illegible] metaprogramming[illegible] metaprogramming [illegible]

18.1 [illegible]

[illegible]SArray[illegible]simple array[illegible]

```
// exprtmpl/sarray1.hpp
#include <stddef.h>
#include <cassert>
template<typename T>
class SArray {
  public:
    // [illegible]
    explicit SArray (size_t s)
      : storage(new T[s]), storage_size(s) {
        init();
    }
    // [illegible]
    SArray (SArray<T> const& orig)
      :            storage(new            T[orig.size()]),
  storage_size(orig.size()) {
        copy(orig);
    }
    // [illegible]: [illegible]
    ~SArray() {
      delete[] storage;
    }
    // [illegible]
    SArray<T>& operator= (SArray<T> const& orig) {
      if (&orig!=this) {
        copy(orig);
      }
      return *this;
```

```
        }
        // [illegible]
        size_t size() const {
            return storage_size;
    }
    // [illegible]
    T operator[] (size_t idx) const {
        return storage[idx];
    }
    T& operator[] (size_t idx) {
        return storage[idx];
    }
protected:
    // [illegible]
    void init() {
        for (size_t idx = 0; idx<size(); ++idx) {
            storage[idx] = T();
        }
    }
    // [illegible]
    void copy (SArray<T> const& orig) {
        assert(size()==orig.size());
        for (size_t idx = 0; idx<size(); ++idx) {
            storage[idx] = orig.storage[idx];
        }
    }
private:
```

```
    T*      storage;          // [illegible]
    size_t storage_size;   // [illegible]
};
```

[illegible]

```
// exprtmpl/sarrayops1.hpp
// [illegible]SArrays[illegible]
template<typename T>
SArray<T> operator+ (SArray<T> const& a, SArray<T>
const& b)
{
    SArray<T> result(a.size());
    for (size_t k = 0; k<a.size(); ++k) {
        result[k] = a[k]+b[k];
    }
    return result;
}
// [illegible]SArray[illegible]
template<typename T>
SArray<T> operator* (SArray<T> const& a, SArray<T>
const& b)
{
    SArray<T> result(a.size());
    for (size_t k = 0; k<a.size(); ++k) {
        result[k] = a[k]*b[k];
    }
    return result;
}
```

```
// [illegible]SArray [illegible]
template<typename T>
SArray<T> operator* (T const& s, SArray<T> const& a)
{
   SArray<T> result(a.size());
   for (size_t k = 0; k<a.size(); ++k) {
      result[k] = s*a[k];
   }
   return result;
}
// [illegible]SArray[illegible]scalar[illegible]
// [illegible]scalar [illegible] SArray[illegible]
// [illegible]SArray [illegible] scalar [illegible]
…
```

[illegible]

```
// exprtmpl/sarray1.cpp
#include "sarray1.hpp"
#include "sarrayops1.hpp"
int main()
{
   SArray<double> x(1000), y(1000);…
   x = 1.2*x + x*y;
}
```

[illegible]

1.[illegible]3[illegible]1 000[illegible]

2.[illegible][illegible]SArray[illegible]6 000[illegible]double[illegible]4 000[illegible]double[illegible]

[illegible]

```
tmp1 = 1.2*x;        // [illegible]1 000[illegible]
tmp1
tmp2 = x*y           // [illegible]1 000[illegible]tmp2
tmp3 = tmp1+tmp2;    // [illegible]1 000[illegible]1 000[illegible]
                     //[illegible]tmp3
x = tmp3;            // 1 000[illegible]1 000[illegible]
```

[illegible]

[illegible]computed assignments[illegible]+=[illegible]*=[illegible]SArray[illegible]

```
// exprtmpl/sarrayops2.hpp
// SArray[illegible]
template<class T>
SArray<T>& SArray<T>::operator+= (SArray<T> const& b)
{
    for (size_t k = 0; k<size(); ++k) {
        (*this)[k] += b[k];
    }
    return *this;
```

```
}
// SArray[illegible]
template<class T>
SArray<T>& SArray<T>::operator*= (SArray<T> const& b)
{
    for (size_t k = 0; k<size(); ++k) {
        (*this)[k] *= b[k];
    }
    return *this;
}
// [illegible]
template<class T>
SArray<T>& SArray<T>::operator*= (T const& s)
{
    for (size_t k = 0; k<size(); ++k) {
        (*this)[k] *= s;
    }
    return *this;
}
```

[illegible]

```
// exprtmpl/sarray2.cpp
#include "sarray2.hpp"
#include "sarrayops1.hpp"
#include "sarrayops2.hpp"
int main()
{
```

```
    SArray<double> x(1000), y(1000);
    …
    // [illegible] x = 1.2*x + x*y
    SArray<double> tmp(x);
    tmp *= y;
    x *= 1.2;
    x += tmp;
}
```

[illegible]“[illegible]”[illegible]

• [illegible]

• [illegible]tmp[illegible]

• [illegible]3[illegible]6 000[illegible]double[illegible]4 000[illegible]double[illegible]

[illegible] [24] [illegible]“[illegible]”[illegible]

```
int main()
{
    SArray<double> x(1000), y(1000);
    …
    for (int idx = 0; idx<x.size(); ++idx) {
        x[idx] = 1.2*x[idx] + x[idx]*y[idx]; [25]
    }
}
```

[illegible]x[idx][illegible]y[idx][illegible]x[idx][illegible]2 000[illegible]1 000[illegible]

18.2

1.2*x + x*y;

1.2*x x 1.2 x*y x y

1.2*x + x*y;

```
A_Add< A_Mult<A_Scalar<double>,Array<double> >,
   A_Mult<Array<double>,Array<double> > >
```

Array A_Scalar A_Add A_Mult 18.1 template-id A_Scalar A_Scalar

[illegible]18.1 [illegible]1.2*x + x*y[illegible]

18.2.1 [illegible]

[illegible]A_Add[illegible]A_Mult[illegible]A_Scalar[illegible]

```
// exprtmpl/exprops1.hpp
#include <stddef.h>
#include <cassert>
// [illegible]class trait template[illegible]//[illegible]“[illegible]”[illegible]“[illegible]”[illegible]“[illegible]”
#include "exprops1a.hpp"
// [illegible]
template <typename T, typename OP1, typename OP2>
class A_Add {
  private:
    typename A_Traits<OP1>::ExprRef op1;    // [illegible]1[illegible]
    typename A_Traits<OP2>::ExprRef op2;    // [illegible]2[illegible]
  public:
    // [illegible]
```

```
    A_Add (OP1 const& a, OP2 const& b)
      : op1(a), op2(b) {
  }
  // □□□□□□□□□
  T operator[] (size_t idx) const {
    return op1[idx] + op2[idx];
  }
  // size□□□□□□□□□□□
  size_t size() const {
    assert (op1.size()==0 || op2.size()==0
      || op1.size()==op2.size());
    return op1.size()!=0 ? op1.size() : op2.size();
  }
};
// □□□□□□□□□□□□□□□
template <typename T, typename OP1, typename OP2>
class A_Mult {
  private:
    typename A_Traits<OP1>::ExprRef op1;    // □1□□□□
    typename A_Traits<OP2>::ExprRef op2;    // □2□□□□
  public:
    // □□□□□□□□□□□□□□□□□□□□
    A_Mult (OP1 const& a, OP2 const& b)
      : op1(a), op2(b) {
    }
    // □□□□□□□□□□
    T operator[] (size_t idx) const {
```

```
            return op1[idx] * op2[idx];
        }
        // size[illegible]
        size_t size() const {
            assert (op1.size()==0 || op2.size()==0
                || op1.size()==op2.size());
            return op1.size()!=0 ? op1.size() : op2.size();
        }
    };
```

[illegible]18.1[illegible]

[illegible]scalar [illegible]scalar[illegible]scalar[illegible]0[illegible]A_Scalar[illegible]

exprtmpl/exprscalar.hpp

```
// [illegible]
template <typename T>
class A_Scalar {
  private:
    T const& s; // scalar[illegible]
  public:
    // [illegible]
    A_Scalar (T const& v)
      : s(v) {
    }
    // [illegible]scalar[illegible]
```

```
        T operator[] (size_t) const {
            return s;
        }
        //scalar□□□□□□□□0
        size_t size() const {
            return 0;
        };
    };
```

□□□□□□□A_Scalar □□□□□□□□□□□□□□□□A_Scalar□□□□□□□□□□□scalar□□□□

□□□□□□□□□□□□□□□A_Traits□□□□□□□

typename A_Traits<OP1>::ExprRef op1;   // □1□□□□

typename A_Traits<OP2>::ExprRef op2;   // □2□□□□

□□A_Scalar□□A_Scalar□□□□□□□□□□□□□□□□□□scalar□□□□“□□□”□□□“□□□□”□□□□□□□□□□□□□□

•□□□□□□□□

OP1 const& op1;   // □□1□□□□□

OP2 const& op2;   // □□2□□□□□

•□□□□scalar□□□□□□

OP1 op1;   // □□□□□□□□1□□□

OP2 op2;   // □□□□□□□□2□□□

□□□trait class□□□□□□□□□□□□□□□□□□□□□□□□□□□□□scalar□□□

```
// exprtmpl/exprops1a.hpp
```

```
/* □□□□□□□□“□□□□□□□”□□□trait class
 * - □□□□□: □□□
 * - □□scalar: □□
 */
template <typename T> class A_Scalar;
// □□□□
template <typename T>
class A_Traits {
  public:
    typedef T const& ExprRef;    // □□□□□□typedef□□□□□
□
};
// □□scalar□□□□□
template <typename T>
class A_Traits<A_Scalar<T> > {
  public:
    typedef A_Scalar<T> ExprRef;   // □□□□□□□□□□□□
};
```

□□□□□□A_Scalar□□□□□□□□□□scalar□□□□□□□□□□□□□□scalar□

18.2.2 Array□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ Array□□□Array□□□□□□□□□□□□□□□□□□□□□□□“□□□□”□□□□□A_Add□□□□□□□□□□□□□□□□□□Array□□□

```
template <typename T, typename Rep = SArray<T> >
```

```
class Array;
```

[illegible]Rep[illegible]SArray [26] [illegible]Array[illegible]template-id[illegible]A_Add[illegible]A_Mult[illegible]Array [illegible]A_Mult [illegible] Rep[illegible]Array[illegible]SArray[illegible]template-id[illegible]

[illegible]SArray[illegible]

```
// exprtmpl/exprarray.hpp
#include <stddef.h>
#include <cassert>
#include "sarray1.hpp"
template <typename T, typename Rep = SArray<T> >
class Array {
  private:
    Rep expr_rep;    // [illegible]
  public:
    // [illegible]
    explicit Array (size_t s)
      : expr_rep(s) {
    }
    // [illegible]
    Array (Rep const& rb)
      : expr_rep(rb) {
    }
    // [illegible]
    Array& operator= (Array const& b) {
```

```
        assert(size()==b.size());
        for (size_t idx = 0; idx<b.size(); ++idx) {
            expr_rep[idx] = b[idx];
    }
    return *this;
}
// [illegible]
template<typename T2, typename Rep2>
Array& operator= (Array<T2, Rep2> const& b) {
    assert(size()==b.size());
    for (size_t idx = 0; idx<b.size(); ++idx) {
        expr_rep[idx] = b[idx];
    }
    return *this;
}
// size[illegible]
size_t size() const {
    return expr_rep.size();
}
// [illegible]
T operator[] (size_t idx) const {
    assert(idx<size());
    return expr_rep[idx];
}
T& operator[] (size_t idx) {
    assert(idx<size());
    return expr_rep[idx];
```

```
        }
        // [illegible]
        Rep const& rep() const {
            return expr_rep;
        }
        Rep& rep() {
            return expr_rep;
        }
    };
```

[illegible]Rep[illegible]Rep[illegible]

18.2.3 [illegible]

[illegible]Array[illegible]A_Add[illegible]+[illegible]

[illegible] 3 [illegible]array-array[illegible]array-scalar[illegible]scalar-array[illegible]

```
// exprtmpl/exprops2.hpp
// [illegible]
template <typename T, typename R1, typename R2>
Array<T,A_Add<T,R1,R2> >
operator+ (Array<T,R1> const& a, Array<T,R2> const& b) {
    return Array<T,A_Add<T,R1,R2> >
        (A_Add<T,R1,R2>(a.rep(),b.rep()));
```

```
}
// □□□□□□
template <typename T, typename R1, typename R2>
Array<T, A_Mult<T,R1,R2> >
operator* (Array<T,R1> const& a, Array<T,R2> const&
b) {
    return Array<T,A_Mult<T,R1,R2> >
        (A_Mult<T,R1,R2>(a.rep(), b.rep()));
}
// scalar□□□□□
template <typename T, typename R2>
Array<T, A_Mult<T,A_Scalar<T>,R2> >
operator* (T const& s, Array<T,R2> const& b) {
    return Array<T,A_Mult<T,A_Scalar<T>,R2> >
        (A_Mult<T,A_Scalar<T>,R2>(A_Scalar<T>(s),
b.rep()));
}
// □□□scalar□□
// scalar□□□□□
// □□□scalar□□
…
```

□□□ A_Add<>□□□□□□□□□□□□□□□

```
A_Add<T,R1,R2>(a.rep(),b.rep())
```

□□□

```
return Array<T,A_Add<T,R1,R2> > (… );
```

scalar A_Scalar A_Mult

```
A_Mult<T,A_Scalar<T>,R2>(A_Scalar<T>(s), b.rep())
```

```
return Array<T,A_Mult<T,A_Scalar<T>,R2> > (… );
```

18.2.4

meta/exprmain.cpp

```
int main()
{
   Array<double> x(1000), y(1000);
   x = 1.2*x + x*y;
   …
}
```

x y Rep SArray<double>

x y “ ”

```
1.2*x + x*y
```

* scalar-array

operator* scalar-array

```
template <typename T, typename R2>
Array<T, A_Mult<T,A_Scalar<T>,R2> >
operator* (T const& s, Array<T,R2> const& b) {
   return Array<T,A_Mult<T,A_Scalar<T>,R2> >
```

```
        (A_Mult<T,A_Scalar<T>,R2>(A_Scalar<T>(s),
    b.rep()));
    }
```

□□□□□□□□□□double□Array<double, SArray<double> >□□□□□□□□□□□□□

```
Array<double, A_Mult<double, A_Scalar<double>, SArray<double> > >
```

□□□□□□□□□□double□1.2□A_Scalar<double>□□□□□□□□□□□x□SArray<double>□□□

□□□□□□□□2□□□□□□□□x*y□□□array-array□□□□□□□□□□□□□□operator*□

```
template <typename T, typename R1, typename R2>
Array<T, A_Mult<T,R1,R2> >
operator* (Array<T,R1> const& a, Array<T,R2> const& b) {
    return Array<T,A_Mult<T,R1,R2> >
        (A_Mult<T,R1,R2>(a.rep(), b.rep()));
}
```

□□□□□□□□□□□Array<double, SArray<double> >□□□□□□□□□

```
Array<double, A_Mult<double, SArray<double>, SArray<double> > >
```

□□□□A_Mult□□□□□□□□□□□□□□□□SArray<double>□□□□□□□□□□x□□□□□□□□□□y□□□

□□□□□+□□□□□□□□□□□□array-array□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□array-array□operator+□

```
template <typename T, typename R1, typename R2>
```

```
Array<T,A_Add<T,R1,R2> >
operator+ (Array<T,R1> const& a, Array<T,R2> const&
b) {
    return Array<T,A_Add<T,R1,R2> >
        (A_Add<T,R1,R2>(a.rep(),b.rep()));
}
```

□□□double□□□T□R1□□□

```
A_Mult<double, A_Scalar<double>, SArray<double> >
```

□□□□□□R2□□□□□

```
A_Mult<double, SArray<double>, SArray<double> >
```

□□□□□□□□□□□□□□□□□□□□□□□

```
Array<double,
    A_Add<double,
        A_Mult<double, A_Scalar<double>, SArray<double>
>,
        A_Mult<double,                    SArray<double>,
SArray<double>>>>
```

□□□□□□Array□□□□□□□□□□□□□□□□

```
template <typename T, typename Rep = SArray<T> >
class Array {
    public:
        ...
        //□□□□□□□□□□□□□□□□
        template<typename T2, typename Rep2>
        Array& operator= (Array<T2, Rep2> const& b) {
            assert(size()==b.size());
            for (size_t idx = 0; idx<b.size(); ++idx) {
```

```
            expr_rep[idx] = b[idx];
        }
        return *this;
    }
    ...
};
```

□□□□□□□□□□□□□Array□□b□□□□□□□□□□□□□□x□□□□□□□□□□□Array□□□□□□□

```
A_Add<double,
    A_Mult<double, A_Scalar<double>, SArray<double> >,
    A_Mult<double, SArray<double>, SArray<double> > > >
```

□□□□□□□□□□□□□□□□□□□□□□□□□□x□□□□□□

```
(1.2*x[idx]) + (x[idx]*y[idx])
```

□□□□□□□□□□□□□□□□

18.2.5 □□□□□□□

□□□□Rep□□□□A_Mult□□A_Add□□□□□□□□□□□□□□□□□□□□□□□□□□□□a+b=c □□

```
x[y] = 2*x[y];
```

□□□□□□□□□

```
for (size_t idx = 0; idx<y.size(); ++idx) {
    x[y[idx]] = 2*x[y[idx]];
}
```

[illegible]A_Mult[illegible]const[illegible]non-const[illegible]

```
// exprtmpl/exprops3.hpp
template<typename T, typename A1, typename A2>
class A_Subscript {
  public:
    //[illegible]
    A_Subscript (A1 const & a, A2 const & b)
      : a1(a), a2(b) {
    }
    // [illegible]
    T operator[] (size_t idx) const {
      return a1[a2[idx]];
    }
    T& operator[] (size_t idx) {
      return a1[a2[idx]];
    }
    // size[illegible]
    size_t size() const {
      return a2.size();
    }
  private:
    A1 const & a1;    // [illegible]1[illegible]
    A2 const & a2;    // [illegible]2[illegible]
};
```

[illegible]Array[illegible][illegible]const[illegible]

```
// exprtmpl/exprops4.hpp
template<typename T, typename R1, typename R2>
Array<T,A_Subscript<T,R1,R2> >
Array<T,R1>::operator[] (Array<T,R2> const & b) {
   return Array<T,A_Subscript<T,R1,R2> >
      (A_Subscript<T,R1,R2>(this—>rep(),b.rep()));
}
```

18.3 [illegible]

[illegible]

[illegible]matrix[illegible]-vector[illegible]

```
x = A*x;
```

[illegible]x[illegible]n[illegible]vector[illegible]A[illegible]n×n[illegible]x[illegible]x[illegible]

```
x = A*y
```

[illegible]x[illegible]y[illegible]

[illegible]Rober Davies[illegible]NewMat[illegible][NewMat][illegible]

[illegible]Jaakko Järvi[illegible] Gary Powell[illegible] Lambda Library [illegible] [Lambdalib][illegible]

```
void lambda_demo (std::vector<long*> & ones) {
   std::sort(ones.begin(), ones.end(), *_1 > *_2);
}
```

[illegible]Lambda[illegible] Lambda [illegible]_1[illegible]_2[illegible]Lambda[illegible]

18.4 [illegible]

[illegible]Todd Veldhuizen [illegible] David Vandevoorde (Todd[illegible])[illegible]C++[illegible] C++[illegible]Copier[illegible]Copier[illegible]CopierInterface[illegible]copy_to[illegible]

```
template<typename T>
class CopierInterface {
   public:
      virtual void copy_to(Array<T, SArray<T> >&) const;
```

```
};
template<typename T, typename X>
class Copier : public CopierInterface<T> {
  public:
    Copier(X const &x): expr(x) {}
    virtual void copy_to(Array<T, SArray<T> >&) const
{
      // [illegible]
      ...
    }
  private:
    X const &expr;
};
template<typename T, typename Rep = SArray<T> >
class Array {
  public:
    // [illegible]
    Array<T, Rep>& operator=(CopierInterface<T>
const &b) {
      b.copy_to(rep);
    };
    ...
};
template<typename T, typename A1, typename A2>
class A_mult {
  public:
    operator Copier<T, A_Mult<T, A1, A2> >();
```

…

};

[illegible]

C++[illegible] valarray [illegible]Array[illegible]valarray[illegible]“[illegible]”[illegible]valarray[illegible]Vandevoorde[illegible]C++[illegible]Array[illegible]valarray[illegible]valarray[illegible]Array[illegible]Rep[illegible]Rep[illegible]foo()[illegible]

```
double foo(Array<double> const&);
```

[illegible] foo(1.2*x)[illegible]Rep[illegible]

```
template<typename Rep>
double foo(Array<double, Rep> const&);
```

[illegible]

[illegible]C++[illegible]valarray[illegible]valarray [illegible] valarray[illegible]

[illegible]STL[illegible][27] [illegible]Borland C++[illegible]4[illegible]

---

[1]. □□□□□□□□□□□□□□□□□□□□□□□□□□□Greek polumorphos□□□□□

[2]. □□□

[3]. □□□□□□□□□□□□□□□□□□[CzarneckiEiseneckerGenPro]□6.5□□6.7□□

[4]. □□C++□□□□□□□□□□□□□[JosuttisStdLib]□□□□□□□□□□□□□□□□□□

[5]. EBCDIC□Extended Binary-Coded Decimal Interchange Code□□□□□□□□IBM□□□□□□□□□IBM□□□□□□□□□

[6]. □□□□□□C++□□□

[7]. □□□□C++□□□13.3□□□

[8]. □□□□□ policy □□□□□□□□□□□□□ Policy □□□□□□□ policy □□□□□□□□□□SumPolicy□□□□□□□□□□□□□

[9]. □□□□□□□□□□□int& &□□□□□□□□□□□□T const□□□□□□□□□int const□□□T□□□□□□□□int const const□□□□□□□□□□□□□□□□□□□□□□□

[10]. □□□□□□□□□□□□□□□volatile□const volatile□□□□□□□□□reference□□□□□□□□□□

[11]. □□□promotion □□□□“□□”□□“□□□□”□□□□“size□□□□□□□□□□□”□

[12]. [illegible]T[illegible]T1[illegible]T2[illegible]

[13]. [illegible]C++[illegible] 13.9 [illegible]

[14]. [illegible]PolicySelector[illegible]4[illegible]Setter[illegible]BreadSlicer<>[illegible]PolicySelector[illegible]4[illegible]

[15]. [illegible]C++[illegible]10.2/6[illegible][Standard 98][illegible][EllisStroustrupARM][illegible]10.1.1[illegible]

[16]. [illegible]16.1[illegible]Discriminator[illegible]22.7[illegible]BaseMem[illegible]

[17]. [illegible]boost.compressed_pair[illegible]

[18]. [illegible]C++[illegible]

[19]. [illegible]Boost Iterator Adaptor Library[illegible]

[20]. [illegible]“[illegible]”[illegible]metaprogramming[illegible]

[21]. [illegible]12.4[illegible]metaprogramming[illegible]

[22]. [illegible]metaprogram[illegible]Fortran[illegible]Fortran[illegible]

[23]. [illegible]Erwin Unruh[illegible][Unruh-PrimeOrig][illegible]

[24]. [illegible]

[25]. [illegible]for[illegible]

[26]. [illegible]Sarray[illegible]Sarray[illegible]

[27]. STL[illegible]C++[illegible]STL[illegible]C++[illegible]

[illegible]4[illegible]

[illegible]C [illegible]“[illegible]”[illegible]C[illegible]

[illegible]

- [illegible]
- [illegible]
- tuple[illegible]
- [illegible]

[illegible]C++ [illegible]C++[illegible]C++[illegible]

[illegible]19[illegible]

[illegible] [1] [illegible]class[illegible]
[illegible]
[illegible]

```
if (TypeT<T>::IsPtrT) {
   ...
}
else if (TypeT<T>::IsClassT) {
   ...
}
```

[illegible]TypeT<T>::IsPtrT[illegible]
[illegible]T[illegible]
[illegible]

19.1 [illegible]

[illegible]
[illegible]

```
// types/type1.hpp
// [illegible]:[illegible] T [illegible]
template <typename T>
class IsFundaT {
   public:
      enum{ Yes = 0, No = 1};
};
// [illegible]
#define MK_FUNDA_TYPE(T)                                   \
   template<> class IsFundaT<T> {                          \
```

```
        public:                                                \
            enum { Yes = 1, No = 0 };                          \
    };
MK_FUNDA_TYPE(void)
MK_FUNDA_TYPE(bool)
MK_FUNDA_TYPE(char)
MK_FUNDA_TYPE(signed char)
MK_FUNDA_TYPE(unsigned char)
MK_FUNDA_TYPE(wchar_t)
MK_FUNDA_TYPE(signed short)
MK_FUNDA_TYPE(unsigned short)
MK_FUNDA_TYPE(signed int)
MK_FUNDA_TYPE(unsigned int)
MK_FUNDA_TYPE(signed long)
MK_FUNDA_TYPE(unsigned long)
#if LONGLONG_EXISTS
    MK_FUNDA_TYPE(signed long long)
    MK_FUNDA_TYPE(unsigned long long)
#endif // LONGLONG_EXISTS
MK_FUNDA_TYPE(float)
MK_FUNDA_TYPE(double)
MK_FUNDA_TYPE(long double)
#undef MK_FUNDA_TYPE
```

□□□□□□□□□□□□□□□□□□□□□□□□□□IsFundaT<T>::Yes □□□□□0(□□false)□

```
template <typename T>
class IsFundaT {
```

```
   public:
      enum{ Yes = 0, No = 1 };
};
```

[illegible]IsFundaT<T >::Yes[illegible] 1[illegible]true[illegible]

```
MK_FUNDA_TYPE(bool)
```

expands to the following:

```
template<> class IsFundaT<bool> {
   public:
      enum{ Yes = 1, No = 0 };
};
```

[illegible]

```
// types/type1test.cpp
#include <iostream>
#include "type1.hpp"
template <typename T>
void test (T const& t)
{
   if (IsFundaT<T>::Yes) {
      std::cout << "T is fundamental type" << std::endl;
   }
   else {
      std::cout << "T is no fundamental type" <<
std::endl;
   }
}
class MyType {
```

```
};
int main()
{
   test(7);
   test(MyType());
}
```

[illegible]:

```
T is fundamental type
T is no fundamental type
```

[illegible] IsIntegralT [illegible] IsFloatingT[illegible]scalar[illegible]

19.2 [illegible]

[illegible]class[illegible] trait [illegible] class [illegible]class[illegible]

```
// types/type2.hpp
template<typename T>
class CompoundT {                // [illegible]
   public:
      enum { IsPtrT = 0, IsRefT = 0, IsArrayT = 0,
         IsFuncT = 0, IsPtrMemT = 0 };
      typedef T BaseT;
      typedef T BottomT;
```

```
        typedef CompoundT<void> ClassT;
    };
```

[illegible]BaseT[illegible]T[illegible]BottomT[illegible]T[illegible]T[illegible]int**[illegible]BaseT[illegible]int*[illegible]BottomT[illegible]int[illegible]BaseT[illegible]ClassT[illegible]T[illegible]int(X::*)()[illegible]BaseT[illegible]int()[illegible]ClassT[illegible]X[illegible]T[illegible]ClassT[illegible]CompoundT<void>[illegible]nonclass[illegible]ClassT[illegible]

[illegible]

```
// types/type3.hpp
template<typename T>
class CompoundT<T&> {          // [illegible]
  public:
    enum { IsPtrT = 0, IsRefT = 1, IsArrayT = 0,
      IsFuncT = 0, IsPtrMemT = 0 };
    typedef T BaseT;
    typedef      typename      CompoundT<T>::BottomT
BottomT;
    typedef CompoundT<void> ClassT;
};
template<typename T>
class CompoundT<T*> {          // [illegible]
  public:
    enum { IsPtrT = 1, IsRefT = 0, IsArrayT = 0,
      IsFuncT = 0, IsPtrMemT = 0 };
    typedef T BaseT;
```

```
    typedef typename CompoundT<T>::BottomT
BottomT;
    typedef CompoundT<void> ClassT;
};
```

[illegible]

```
// types/type4.hpp
#include <stddef.h>
template<typename T, size_t N>
class CompoundT <T[N]> {    // [illegible]
  public:
    enum { IsPtrT = 0, IsRefT = 0, IsArrayT = 1,
      IsFuncT = 0, IsPtrMemT = 0 };
    typedef T BaseT;
    typedef typename CompoundT<T>::BottomT
BottomT;
    typedef CompoundT<void> ClassT;
};
template<typename T>
class CompoundT <T[]> {    //   [illegible]
  public:
    enum { IsPtrT = 0, IsRefT = 0, IsArrayT = 1,
      IsFuncT = 0, IsPtrMemT = 0 };
    typedef T BaseT;
    typedef typename CompoundT<T>::BottomT
BottomT;
    typedef CompoundT<void> ClassT;
```

```
};
template<typename T, typename C>
class CompoundT <T C::*> {   // [illegible]
  public:
    enum { IsPtrT = 0, IsRefT = 0, IsArrayT = 0,
      IsFuncT = 0, IsPtrMemT = 1 };
    typedef T BaseT;
    typedef typename CompoundT<T>::BottomT BottomT;
    typedef C ClassT;
};
```

[illegible]BottomT[illegible]T[illegible]CompoundT[illegible]T[illegible]T[illegible]

[illegible]

19.3 [illegible]

[illegible]

```
// types/type5.hpp
template<typename R>
class CompoundT<R()> {
  public:
```

```
        enum { IsPtrT = 0, IsRefT = 0, IsArrayT = 0,
           IsFuncT = 1, IsPtrMemT = 0 };
        typedef R BaseT();
        typedef R BottomT();
        typedef CompoundT<void> ClassT;
};
template<typename R, typename P1>
class CompoundT<R(P1)> {
    public:
        enum { IsPtrT = 0, IsRefT = 0, IsArrayT = 0,
           IsFuncT = 1, IsPtrMemT = 0 };
        typedef R BaseT(P1);
        typedef R BottomT(P1);
        typedef CompoundT<void> ClassT;
};
template<typename R, typename P1>
class CompoundT<R(P1, ...)> {
    public:
        enum { IsPtrT = 0, IsRefT = 0, IsArrayT = 0,
           IsFuncT = 1, IsPtrMemT = 0 };
        typedef R BaseT(P1);
        typedef R BottomT(P1);
    typedef CompoundT<void> ClassT;
};
…
```

□□□□□□□□□□□□□□□□□□□□□□□□typedef□□□

[illegible] 8.3.1 [illegible] SFINAE [illegible]test[illegible]U[illegible]SFINAE[illegible]SFINAE [illegible]void[illegible]

```
template<typename T>
class IsFunctionT {
  private:
    typedef char One;
    typedef struct { char a[2]; } Two;
    template<typename U> static One test(...);
    template<typename U> static Two test(U (*)[1]);
  public:
    enum { Yes = sizeof(IsFunctionT<T>::test<T>(0))
== 1 };
    enum { No = !Yes };
};
```

[illegible]IsFunctionT::Yes[illegible]1[illegible]void[illegible]void[illegible]

```
template<typename T>
class IsFunctionT<T&> {
  public:
    enum { Yes = 0 };
```

```
        enum { No = !Yes };
};
template<>
class IsFunctionT<void> {
    public:
        enum { Yes = 0 };
    enum { No = !Yes };
};
template<>
class IsFunctionT<void const> {
    public:
        enum { Yes = 0 };
        enum { No = !Yes };
};
...
```

[illegible]F&[illegible]F*[illegible]F[illegible]

[illegible]CompoundT[illegible]

```
// types/type6.hpp
template<typename T>
class IsFunctionT {
    private:
        typedef char One;
        typedef struct { char a[2]; } Two;
        template<typename U> static One test(...);
        template<typename U> static Two test(U (*)[1]);
    public:
```

```
        enum  {  Yes  =  sizeof(IsFunctionT<T>::test<T>(0))
== 1 };
        enum { No = !Yes };
};
template<typename T>
class IsFunctionT<T&> {
    public:
        enum { Yes = 0 };
        enum { No = !Yes };
};
template<>
class IsFunctionT<void> {
    public:
        enum { Yes = 0 };
        enum { No = !Yes };
};
template<>
class IsFunctionT<void const> {
    public:
        enum { Yes = 0 };
        enum { No = !Yes };
};
// □□ void volatile□void const volatile□□□□□□
...
template<typename T>
class CompoundT {                //□□□□
    public:
```

```
        enum { IsPtrT = 0, IsRefT = 0, IsArrayT = 0,
          IsFuncT = IsFunctionT<T>::Yes,
          IsPtrMemT = 0 };
        typedef T BaseT;
        typedef T BottomT;
        typedef CompoundT<void> ClassT;
    };
```

[illegible]

[illegible]C++[illegible]C++[illegible]

```
template<class T>
struct X {
  long aligner;
  T m;
};
```

[illegible] C++[illegible] X::m [illegible]X::m()[illegible]T[illegible]X<T>[illegible]X0[illegible]

```
struct X0 {
  long aligner;
};
```

[illegible]T[illegible]X<T>[illegible]X0[illegible]aligner[illegible]char[illegible]

[illegible]class[illegible] CompoundT [illegible]

[illegible]class[illegible]
class[illegible]

19.4 [illegible]

[illegible]

[illegible]

```
// types/type7.hpp
struct SizeOverOne { char c[2]; };
template<typename T,
   bool convert_possible = !CompoundT<T>::IsFuncT &&
                           !CompoundT<T>::IsArrayT>
class ConsumeUDC {
   public:
      operator T() const;
};
// [illegible]
template <typename T>
class ConsumeUDC<T, false> {
};
// [illegible]void[illegible]
template <bool convert_possible>
class ConsumeUDC<void, convert_possible> {
};
```

```
char enum_check(bool);
char enum_check(char);
char enum_check(signed char);
char enum_check(unsigned char);
char enum_check(wchar_t);
char enum_check(signed short);
char enum_check(unsigned short);
char enum_check(signed int);
char enum_check(unsigned int);
char enum_check(signed long);
char enum_check(unsigned long);
#if LONGLONG_EXISTS
   char enum_check(signed long long);
   char enum_check(unsigned long long);
#endif // LONGLONG_EXISTS
// □□□float□int□□□□□
char enum_check(float);
char enum_check(double);
char enum_check(long double);
SizeOverOne enum_check(...); // □□□□□□□□□
template<typename T>
class IsEnumT {
   public:
      enum { Yes = IsFundaT<T>::No &&
               !CompoundT<T>::IsRefT &&
               !CompoundT<T>::IsPtrT &&
               !CompoundT<T>::IsPtrMemT &&
```

```
            sizeof(enum_check(ConsumeUDC<T>()))==1
        };
    enum { No = !Yes };
};
```

□□□□□□□□□□□sizeof□□□□□□□□□□□□□□□□□ sizeof □□□□□□□□□□□□□□□□□□□□□□□□□□□□enum_check()□□□enum_check()□□□□char□□□□□1□□□□□□□□□□□□□□□□□□□□enum_check(...) □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□enum_check()□□□□□□□□□□□□□□□□□□□□□□□□□□□SizeOverOne□ [2] □

□□□enum_check□□□□□□□□□□□□□□□□□□□T□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□T□□□□□□□□□□□□□□□T□□□□sizeof□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□class□T□□□□□□□□□□□□□□enum_check()□□□□□□class□□□□□□□□□□□□□□□□UDC□□□□□□□□□□□□□□□□□□□□□□□□□ConsumeUDC□□□□□□□□□□T□□□□□□□□□□□□□sizeof□□□□□□□□□T□□□□□□□□□□□□□□□□□□□□□□□enum_check()□□□□□□□□□□□□□□□□□B□□

- □□□□□□□□□ConsumeUDC<T>□□□
- □□T□□□□□□□□□□□□□□ConsumeUDC□□□□□□□□□□□enum_check□□□□□enum_check()□T□□□
- □□T□□□□□□□□□□□□□ConsumeUDC□□□□□□□□□□□□T□□□□□□□□□□□□□□□□□□□□□□□□□□□□enum_check()□□□□□□□enum_check(int)□ [3] □
- □□T□□class□□□□□□□class□□□

ConsumeUDC<T>□T□□□□□□□□□□□□□□□□□□□□□□□□□□□□
enum_check()□□□□class□□□□□□□□□□□□

•□□□□□□□□□□ T □□□□□□□□□□□□ enum_check()□□□□□□
□□

□□□
IsFundaT□CompoundT□□□□□□□□□□IsEnumT<T>::Yes□□□□□
□□□□□□□□□□□□□IsEnumT::Yes□□1□

19.5 □□cllass□□

□□□□□□□□□□□□□□□□□□□□□□□class□□□□□class□struct□
union□□□□□□□□□□□□□□□15.2.2□□□□□SFINAE□□□□□□□□□

□□□
□□□□class□□□□□□□□□□□□□□□□□□□□□□

```
// types/type8.hpp
template<typename T>
class IsClassT {
  public:
    enum { Yes = IsFundaT<T>::No &&
              IsEnumT<T>::No &&
              !CompoundT<T>::IsPtrT &&
              !CompoundT<T>::IsRefT &&
              !CompoundT<T>::IsArrayT &&
              !CompoundT<T>::IsPtrMemT &&
              !CompoundT<T>::IsFuncT };
    enum { No = !Yes };
};
```

19.6 [illegible]

[illegible]

```
// types/typet.hpp
#ifndef TYPET_HPP
#define TYPET_HPP
// define IsFundaT<>
#include "type1.hpp"
// [illegible] CompoundT<> ([illegible])
//#include "type2.hpp"
// [illegible] CompoundT<> ([illegible]2[illegible])
#include "type6.hpp"
// define CompoundT<> [illegible]#include "type3.hpp"
#include "type4.hpp"
#include "type5.hpp"
// [illegible] IsEnumT<>
#include "type7.hpp"
// [illegible] IsClassT<>
#include "type8.hpp"
// [illegible]
template <typename T>
class TypeT {
  public:
    enum { IsFundaT = IsFundaT<T>::Yes,
           IsPtrT = CompoundT<T>::IsPtrT,
```

```
              IsRefT = CompoundT<T>::IsRefT,
              IsArrayT = CompoundT<T>::IsArrayT,
              IsFuncT = CompoundT<T>::IsFuncT,
              IsPtrMemT = CompoundT<T>::IsPtrMemT,
              IsEnumT = IsEnumT<T>::Yes,
              IsClassT = IsClassT<T>::Yes };
};
#endif // TYPET_HPP
```

[illegible]

```
// types/types.cpp
#include "typet.hpp"
#include <iostream>
class MyClass {
};
void myfunc(){
}
enum E { e1 };
// [illegible]
template <typename T>
void check()
{
   if (TypeT<T>::IsFundaT) {
      std::cout << " IsFundaT ";
   }
   if (TypeT<T>::IsPtrT) {
      std::cout << " IsPtrT ";
   }
```

```
    if (TypeT<T>::IsRefT) {
        std::cout << " IsRefT ";
    }
    if (TypeT<T>::IsArrayT) {
        std::cout << " IsArrayT ";
    }
    if (TypeT<T>::IsFuncT) {
        std::cout << " IsFuncT ";
    }
    if (TypeT<T>::IsPtrMemT) {
        std::cout << " IsPtrMemT ";
    }
    if (TypeT<T>::IsEnumT) {
        std::cout << " IsEnumT ";
    }
    if (TypeT<T>::IsClassT) {
        std::cout << " IsClassT ";
    }
    std::cout << std::endl;
}
// [illegible]
template <typename T>
void checkT (T)
{
    check<T>();
    // [illegible]
    if (TypeT<T>::IsPtrT || TypeT<T>::IsPtrMemT) {
```

```
        check<typename CompoundT<T>::BaseT>();
    }
}
int main()
{
    std::cout << "int:" << std::endl;
    check<int>();
    std::cout << "int&:" << std::endl;
    check<int&>();
    std::cout << "char[42]:" << std::endl;
    check<char[42]>();
    std::cout << "MyClass:" << std::endl;
    check<MyClass>();
    std::cout << "ptr to enum:" << std::endl;
    E* ptr = 0;
    checkT(ptr);
    std::cout << "42:" << std::endl;
    checkT(42);
    std::cout << "myfunc():" << std::endl;
    checkT(myfunc);
    std::cout << "memptr to array:" << std::endl;
    char (MyClass::* memptr) [] = 0;
    checkT(memptr);
}
```

□□□□□□□:

```
int:
    IsFundaT
```

int&:

IsRefT

char[42]:

IsArrayT

MyClass:

IsClassT

ptr to enum:

IsPtrT

IsEnumT

42:

IsFundaT

myfunc():

IsPtrT

IsFuncT

memptr to array:

IsPtrMemT

IsArrayT

19.7

reflection metaprogramming 17

20 90 STL _ _type_traits SGI Silicon Graphics SGI POD STL

[illegible]SGI[illegible]SGI[illegible]_
_type_traits[illegible]_ _type_traits[illegible]

[illegible]SFINAE[illegible]Anderei Alexandrescu[illegible]
sizeof[illegible]

[illegible]Boost[illegible][BoostTypeTraits][illegible]C++[illegible]13.10[illegible]

[illegible]20[illegible] [illegible]

[illegible]C++[illegible]raw
memory[illegible]

[illegible]C++[illegible] -> [illegible] *[illegible]policy[illegible]

[illegible]——[illegible]

• [illegible]policy[illegible]

• [illegible]

[illegible]“[illegible]”[illegible]22[illegible]

20.1 hollder□trulle

□□□□□□□□□□□□□□holder□□□□□□□□□□trule□□□□□□□□□□□□holder□□□□□□holder□

20.1.1 □□□□□□

□C++□□□C++□□□□□□C++□□□□□programming authors□□

```
void do_something()
{
    Something* ptr = new Something;
    // □*ptr□□□□□□
    ptr->perform();
    …
    delete ptr;
}
```

□□□□□new□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□delete □□□

```
void do_something()
{
    Something* ptr = 0;
    try {
        ptr = new Something;
        //□*ptr□□□□□□
```

```
        ptr->perform();
        ...
    }
    catch (...) {
        delete ptr;
        throw;   // [illegible]
    }
    delete ptr;
}
```

[illegible]

```
void do_two_things()
{
    Something* first = new Something;
    first->perform();
    Something* second = new Something;
    second->perform();
    delete second;
    delete first;
}
```

[illegible]

```
void do_two_things()
{
    Something* first = 0;
    Something* second = 0;
```

```
    try {
        first = new Something;
        first->perform();
        second = new Something;
        second->perform();
    }
    catch (...) {
        delete first;
        delete second;
        throw;   //[illegible]
    }
    delete second;
    delete first;
}
```

[illegible]delete[illegible] [4] [illegible]

[illegible]——[illegible]

20.1.2 holder

[illegible]2[illegible]policy[illegible]

[illegible] holder[illegible]

[illegible]hold[illegible]

```
// pointers/holder.hpp
teplate <typename T>
class Holder {
    private:
```

```
    T* ptr;    // [illegible]
public:
    // [illegible]: [illegible]holder[illegible]
    Holder() : ptr(0) {}
    // [illegible]: [illegible]holder[illegible]
    explicit Holder (T* p) : ptr(p) {
    }
    // [illegible]
    ~Holder() {
        delete ptr;
    }
    // [illegible]
    Holder<T>& operator= (T* p) {
        delete ptr;
        ptr = p;
        return *this;
    }
    // [illegible]
    T& operator* () const {
        return *ptr;
    }
    T* operator-> () const {
        return ptr;
    }
    // [illegible]
T* get() const {
    return ptr;
```

```
    }
    // [illegible]
    void release() {
      ptr = 0;
    }
    // [illegible]holder[illegible]
    void exchange_with (Holder<T>& h) {
      swap(ptr,h.ptr);
    }
    // [illegible]
    void exchange_with (T*& p) {
      swap(ptr,p);
    }
  private:
    //[illegible]
    Holder (Holder<T> const&);
    Holder<T>& operator= (Holder<T> const&);
  };
```

[illegible]holder[illegible]ptr[illegible]new[illegible]holder[illegible]delete[illegible] [5] [illegible]release()[illegible]holder[illegible]holder[illegible]holder[illegible]holder[illegible]exchange_with()[illegible]holder[illegible]

[illegible]

```
  void do_two_things()
  {
```

```
    Holder<Something> first(new Something);
    first->perform();
    Holder<Something> second(new Something);
    second->perform();
}
```

□□□□□□□□□□□□□□Holder□□□□□□□□□□□□□□□□□□□□
□□

□□□□□□□□□□□□□□□□□□□Holder□□□

```
Holder<Something> first = new Something;    // □□
```

□□□□□□□ explicit □□□□□□□□□□□□□□□□□□□□□□□□□

```
X x;
Y y(x);    // □□□□
```

□

```
X x;
Y y = x;    // □□□□
```

□□□□□□X□□Y□□□□□□□□□□□□□Y□□□□□□□□□□X□Y□□□□□□□□□□□□Y□□□□□□□□□□□□□□□□□explicit□□□□□□□□□□□□□□□□□□□□□□□□

20.1.3 □□□□holder

□□□□□□□□holder□□□□□□□□□□□□□□holder□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□holder□□□□□□□□□□□□holder □□holder□□□□□□□□□□

```
// pointers/refmem1.hpp
```

```
class RefMembers {
  private:
    MemType* ptr1;        // □□□□□□
    MemType* ptr2;
  public:
    // □□□□□□
    // - □□□2□new□□□□□□□□□□□□□□□□□
    RefMembers ()
    : ptr1(new MemType), ptr2(new MemType) {
  }
  // □□□□□□
  // - □□□2□new□□□□□□□□□□□□□□□
  RefMembers (RefMembers const& x)
    :     ptr1(new     MemType(*x.ptr1)),     ptr2(new
MemType(*x.ptr2)) {
  }
  // □□□□□
  const RefMembers& operator= (RefMembers const& x)
{
    *ptr1 = *x.ptr1;
    *ptr2 = *x.ptr2;
    return *this;
  }
  ~RefMembers () {
    delete ptr1;
    delete ptr2;
  }
```

```
   ...
};
```

[illegible]holder[illegible]

```
// pointers/refmem2.hpp
#include "holder.hpp"
class RefMembers {
   private:
      Holder<MemType> ptr1;         // [illegible]
      Holder<MemType> ptr2;
   public:
      // [illegible]
      // - [illegible]
      RefMembers ()
         : ptr1(new MemType), ptr2(new MemType) {
      }
      // [illegible]
      // - [illegible]
      RefMembers (RefMembers const& x)
         :    ptr1(new    MemType(*x.ptr1)),    ptr2(new
   MemType(*x.ptr2)) {
      }
      // [illegible]
   const RefMembers& operator= (RefMembers const& x)
{
      *ptr1 = *x.ptr1;
      *ptr2 = *x.ptr2;
      return *this;
```

```
    }
    // [illegible]
    // ([illegible]ptr1[illegible]ptr2[illegible])
    ...
};
```

[illegible]

20.1.4 [illegible]

holder[illegible]“[illegible]” [6] [illegible]RAII[illegible] [StroustrupDnE][illegible] policy [illegible]

```
void do_something()
{
    // [illegible]
    RES1* res1 = acquire_resource_1();
    RES2* res2 = acquire_resource_2();
    ...
    // [illegible]
    release_resource_2(res);
    release_resource_1(res);
}
```

[illegible]

```
void do_something ()
{
    // [illegible]
    Holder<RES1,...> res1(acquire_resource_1());
    Holder<RES2,...> res2(acquire_resource_2());
```

```
    ...
}
```

□□□□□□□□□□□□□□□□□□□□holder□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

20.1.5 holder□□□

holder□□□□□□□□□□□□□□□□□□□□□□□□

```
Something* load_something()
{
    Something* result = new Something;
    read_something(result);
    return result;
}
```

□□□□□□□□□□□□□□□□□□□

1.□□□□□□□□□□read_something()□□□□□□□□□□□□□□□□□□□□□

2.load_something()□□□□□□□□□□□□□□

□□□□□□□□□□□holder□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
Something* load_something()
{
    Holder<Something> result(new Something);
    read_something(result.get_pointer());
    Something* ret = result.get_pointer();
    result.release();
    return ret;
}
```

□□□□□□□□□□read_something()□□□□holder□□□□□□□□□□□□□□□□□□□□□□□get_pointer()□□

holder [illegible] result.get_pointer()[illegible] load_something()[illegible] holder[illegible] result.get_pointer()[illegible] ret[illegible]

[illegible] get_pointer()[illegible] * [illegible] & [illegible] -> [illegible]

```
read_something(&*result);
read_something(result.operator->());
```

[illegible]

[illegible] release()[illegible]

```
Something* ret = result.get_pointer();
result.release();
return ret;
```

[illegible] release()[illegible]

```
template <typename T>
class Holder {
   ...
   T* release() {
      T* ret = ptr;
      ptr = 0;
      return ret;
   }
```

```
    ...
};
```

[illegible]

```
return result.release();
```

[illegible] policy [illegible] holder [illegible]

20.1.6 [illegible] holder

[illegible] holder [illegible] holder [illegible] holder [illegible] holder [illegible] holder [illegible] holder [illegible] holder [illegible] holder [illegible]

[illegible]

```
Holder<Something> h1(new Something);
Holder<Something> h2(h1.release());
```

[illegible]

```
Holder<X> h = p;
```

[illegible] explicit [illegible]

```
Holder<Something> h2 = h1.release();   // [illegible]
```

20.1.7 [illegible] holder

[illegible] holder [illegible] "[illegible]" [illegible] holder [illegible] "[illegible]" [illegible]

```
MyClass x;
callee(h1.release(),x);   // [illegible]x[illegible]!
```

[illegible]h1.release()[illegible]x[illegible]x[illegible]h1[illegible]holder[illegible]

[illegible] holder [illegible] holder [illegible]holder [illegible] release() [illegible] load_something()[illegible]

```
Something* creator()
{
   Holder<Something> h(new Something);
   MyClass x;   // [illegible]
   return h.release();
}
```

[illegible]h[illegible]h[illegible]x[illegible]"[illegible]"[illegible]

[20.1.8 trule]

[illegible]holder[illegible]trule[illegible]transfer capsule[illegible]

```
// pointers/trule.hpp
#ifndef TRULE_HPP
#define TRULE_HPP
```

```
template <typename T>
class Holder;
template <typename T>
class Trule {
   private:
      T* ptr;    // trule□□□□□□(□□□□□)
   public:
      // □□□□□□□trule□□□□□□□□□□□□holder
         // □□□□□□□□□□□□□
      Trule (Holder<T>& h) {
         ptr = h.get();
         h.release();
      }
      // □□□□□□
      Trule (Trule<T> const& t) {
         ptr = t.ptr;
         const_cast<Trule<T>&>(t).ptr = 0;
      }
      // □□□□
      ~Trule() {
         delete ptr;
      }
   private:
      Trule(Trule<T>&);                    // □□□trule□□□□
      Trule<T>& operator= (Trule<T>&); // □□□□□□
      friend class Holder<T>;
};
```

```
#endif // TRULE_HPP
```

[illegible]trule[illegible]holders[illegible]trule[illegible]rvalues[illegible]reference-to-const[illegible]Trule[illegible]trule[illegible]non-const[illegible]const[illegible]non-const[illegible]holder[illegible]trule[illegible]trule<T>[illegible]trule<T> const[illegible]

[illegible]holder[illegible]trule[illegible]holder [illegible]trule[illegible]holder[illegible]

[illegible]trule[illegible]holder[illegible] non-const [illegible]Trule[illegible]trule[illegible]

[illegible]trule[illegible]holder [illegible]holder[illegible]

```
// pointers/holder2extr.hpp
template <typename T>
class Holder {
  // [illegible]
  ...
  public:
    Holder (Trule<T> const& t) {
```

```
            ptr = t.ptr;
            const_cast<Trule<T>&>(t).ptr = 0;
        }
        Holder<T>& operator= (Trule<T> const& t) {
            delete ptr;
            ptr = t.ptr;
        const_cast<Trule<T>&>(t).ptr = 0;
        return *this;
    }
};
```

□□□□□□□holder/trule□□□□□□□□□□□□□□load_something()□□□□□□□□□□load_something()□main□□□

```
// pointers/truletest.cpp
#include "holder2.hpp"
#include "trule.hpp"
class Something {
};
void read_something (Something* x)
{
}
Trule<Something> load_something()
{
    Holder<Something> result(new Something);
    read_something(result.get());
    return result;
}
int main()
```

```
{
   Holder<Something> ptr(load_something());
   ...
}
```

[illegible]

20.2 [illegible]

holder[illegible]Trule[illegible]

[illegible]policy[illegible]policy[illegible]policy[illegible]

[illegible]0[illegible]C++[illegible]

```
template <typename T ... >
class CountingPtr {
  public:
    // [illegible]T[illegible]
    explicit CountingPtr (T*);
    // [illegible]:
```

```
        CountingPtr (CountingPtr<T… > const&);
        // [illegible]:
        inline ~CountingPtr();
        // [illegible]
    // [illegible]
      // ([illegible]):
      CountingPtr<T…  >&  operator=  (CountingPtr<T…  >
const&);
      // [illegible]:
      inline T& operator* ();
      inline T* operator-> ();
      …
    };
```

[illegible]T[illegible]CountingPtr[illegible]policy[illegible]policy[illegible]15[illegible]

[illegible] CountingPtr [illegible]0[illegible]

20.2.1 [illegible]

[illegible]

[illegible]C++[illegible]::operator new[illegible]::operator new[illegible]

[illegible]

[illegible]policy[illegible]policy[illegible]15[illegible]policy[illegible]

20.2.2 [illegible]

[illegible] 0[illegible]

[illegible]

```
class CounterPolicy {
  public:
    // [illegible]4[illegible] ([illegible])[illegible]
    //[illegible]
    CounterPolicy();
    CounterPolicy(CounterPolicy const&);
    ~CounterPolicy();
    CounterPolicy& operator=(CounterPolicy const&);
  // [illegible]T[illegible]
  void init(T*);          // [illegible]1[illegible]
  void dispose(T*);       // [illegible]
  void increment(T*);     // [illegible]1[illegible]
  void decrement(T*);     // [illegible]1[illegible]
```

```
        bool is_zero(T*);      // [illegible]0
        ...
    };
```

[illegible] T [illegible] CountingPtr [illegible]policy[illegible]

[illegible]CountingPtr[illegible]
CountingPtr[illegible]

20.2.3 [illegible]

[illegible]C++[illegible]delete[illegible] C [illegible]
free()[illegible]delete[][illegible]

[illegible]policy[illegible]policy[illegible]

```
    class ObjectPolicy {
        public:
            // [illegible]4[illegible] ([illegible])[illegible]
            //[illegible]
            ObjectPolicy();
            ObjectPolicy(CounterPolicy const&);
            ~ObjectPolicy();
            ObjectPolicy& operator=(ObjectPolicy const&);
            //[illegible]T[illegible]
            void dispose (T*);
    };
```

[illegible]operator* [illegible] operator-> [illegible]policy[illegible]policy [illegible]

[illegible]CountingPtr[illegible]policy[illegible]

```
// pointers/stdobjpolicy.hpp
class StandardObjectPolicy {
  public:
    template<typename T> void dispose (T* object) {
      delete object;
    }
};
```

[illegible]new[][illegible]policy[illegible]

```
// pointers/stdarraypolicy.hpp
class StandardArrayPolicy {
  public:
    template<typename T> void dispose (T* array) {
      delete[] array;
    }
};
```

[illegible]dispose()[illegible]policy[illegible]15.1.6[illegible]

20.2.4 CountingPtr [illegible]

[illegible]policy[illegible]CountingPtr[illegible]

```
// pointers/countingptr.hpp
template<typename T,
         typename CounterPolicy = SimpleReferenceCount,
         typename ObjectPolicy = StandardObjectPolicy>
class CountingPtr : private CounterPolicy, private ObjectPolicy {
    private:
      // typedef□□□□□□□:
      typedef CounterPolicy CP;
      typedef ObjectPolicy   OP;
      T* object_pointed_to;       // □□□□□□
//(□□□□□□□□□□□□□NULL)
public:
    // □□□□□□ (□□□□□□□□□□□□□explicit□□□):
    CountingPtr() {
      this->object_pointed_to = NULL;
    }
    // □□□□□□□□□□□ (□□□□□□□□□□):
    explicit CountingPtr (T* p) {
      this->init(p);            // □□□□□□□□□
    }
    // □□□□□□:
    CountingPtr (CountingPtr<T,CP,OP> const& cp)
      : CP((CP const&)cp),       // □□ policy
        OP((OP const&)cp) {
          this->attach(cp);      // □□□□□□□□□□□□
    }
```

```
// □□□□:
~CountingPtr() {
   this->detach();          // □□□□□
                      // (□□□□□□0□□□□□□□)
}
// □□□□□□□□□□□□
CountingPtr<T,CP,OP>& operator= (T* p) {
   // □□□□□□□□ *p :
   assert(p != this->object_pointed_to);
   this->detach();          // □□□□□
                      // (□□□□□□0□□□□□□□)
   this->init(p);          // □□□□□□□□□□□□
   return *this;
}
// □□□□□□□ (□□□□□□□□□□□):
CountingPtr<T,CP,OP>&
operator= (CountingPtr<T,CP,OP> const& cp) {
   if (this->object_pointed_to != cp.object_pointed_to) {
      this->detach();   // □□□□□
                     // (□□□□□□0□□□□□□□)
      CP::operator=((CP const&)cp);   // □policy□□□□
      OP::operator=((OP const&)cp);
      this->attach(cp);   // □□□□□□□□□□□
   }
   return *this;
}
// □□□□□□□□□□□□:
```

```
    T* operator-> () const {
        return this->object_pointed_to;
    }
    T& operator* () const {
        return *this->object_pointed_to;
    }
    // □□□□□□□□□□□□□□□□□□
    ...
private:
    // □□□□:
    // - □□□□□□□□□□ (□□□□□□□□□)
    void init (T* p) {
        if (p != NULL) {
            CounterPolicy::init(p);
        }
        this->object_pointed_to = p;
    }
    // - □□□□□□□□□□□ (□□□□□□□)
    void attach (CountingPtr<T,CP,OP> const& cp) {
        this->object_pointed_to = cp.object_pointed_to;
        if (cp.object_pointed_to != NULL) {
            CounterPolicy::increment(cp.object_pointed_to);
        }
    }
    // - □□□□□ (□□□□□□0□□□□□□□)
    void detach() {
        if (this->object_pointed_to != NULL) {
```

```
        CounterPolicy::decrement(this-
    >object_pointed_to);
        if  (CounterPolicy::is_zero(this->object_pointed_to))
    {
          // [illegible]:
          CounterPolicy::dispose(this->object_pointed_to);
          // [illegible]object policy[illegible]:
          ObjectPolicy::dispose(this->object_pointed_to);
        }
      }
    }
  };
```

[illegible] 0[illegible]

[illegible]policy[illegible]CountingPtr[illegible]policy[illegible]policy[illegible]

[illegible]policy[illegible]policy[illegible] 16.2 [illegible]16.2.2[illegible]BaseMemberPair[illegible]policy[illegible]BaseMemberPair[illegible]

[illegible]16.1[illegible]

20.2.5 [illegible]

[illegible]CountingPtr[illegible]policy[illegible]policy[illegible] policy[illegible]policy[illegible]policy[illegible]

[illegible]CountingPtr[illegible]C++[illegible][7][illegible]alloc_counter()[illegible]dealloc_counter()[illegible]size_t[illegible]

```
// pointers/simplerefcount.hpp
#include <stddef.h>   // [illegible] size_t [illegible]
#include "allocator.hpp"
class SimpleReferenceCount {
  private:
    size_t* counter;    // [illegible]
  public:
    SimpleReferenceCount () {
      counter = NULL;
    }
    // [illegible]
    // [illegible]
  public:
    // [illegible]1:
    template<typename T> void init (T*) {
      counter = alloc_counter();
      *counter = 1;
```

```
        }
        // [illegible]:
        template<typename T> void dispose (T*) {
            dealloc_counter(counter);
        }
        // [illegible]1:
        template<typename T> void increment (T*) {
            ++*counter;
        }
        // [illegible]1:
        template<typename T> void decrement (T*) {
            --*counter;
        }
        // [illegible]0:
        template<typename T> bool is_zero (T*) {
            return *counter == 0;
        }
    };
```

[illegible] policy [illegible] CountingPtr[illegible] CountingPtr[illegible]policy[illegible]

[illegible]policy[illegible]T[illegible]policy[illegible]

20.2.6 [illegible]

[illegible]policy[illegible]policy[illegible]policy[illegible]

[illegible]policy[illegible]

```
// pointers/memberrefcount.hpp
template<typename ObjectT,       // [illegible]
         typename CountT,        // [illegible]
         CountT ObjectT::*CountP> // [illegible]
class MemberReferenceCount
{
  public:
    // [illegible]
    // [illegible]1:
    void init (ObjectT* object) {
      object->*CountP = 1;
    }
    // [illegible]:
    void dispose (ObjectT*) {
    }
    // [illegible]1:
    void increment (ObjectT* object) {
      ++object->*CountP;
    }
    // [illegible]1:
```

```
            void decrement (ObjectT* object) {
                --object->*CountP;
            }
        // □□□□□□□□0:
        template<typename T> bool is_zero (ObjectT* object)
    {
            return object->*CountP == 0;
        }
    };
```

□□□□□□policy□□□

```
    class ManagedType {
        private:
            size_t ref_count;
        public:
            typedef CountingPtr<ManagedType,
                        MemberReferenceCount
                            <ManagedType,
                                size_t,
                                &ManagedType::ref_count> >
                    Ptr;
            ...
    };
```

□□□□□□□□□□□□□□□□□□□ManagedType::Ptr□□□□□□□“□□□□□□ManagedType□□□□”□□□□□□□□□□□□□□□□□□CountingPtr□□

20.2.7 □□□

[illegible]C++[illegible]X const* [illegible]X* const[illegible]

[illegible] X const* [illegible] CountingPtr<X const>[illegible]X* const[illegible]CountingPtr<X> const[illegible]CountingPtr[illegible]

[illegible] const [illegible]T[illegible]T[illegible]const[illegible]

[illegible]int* [illegible]int const* [illegible] const [illegible]CountingPtr<int>[illegible] CountingPtr<int const>[illegible] int const*[illegible]not const-qualified[illegible]const[illegible]T[illegible] T const* [illegible]

[illegible]

20.2.8 [illegible]

[illegible]

- [illegible]void* [illegible]
- [illegible]
- [illegible]bool[illegible]false[illegible]true[illegible]

[illegible]CountingPtr[illegible]CountingPtr<int const>[illegible]int const*[illegible]

[illegible]CountingPtr[illegible]

CountingPtr<X>□□□□□□□□□□void*□X*□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□cp□□□□□□□□

•delete cp□□□::delete cp□□□□□□□□□

•□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□, cp[n]□cp2 - cp1□□□

□□□□□□□□□

□□□□□□CountingPtr□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
CountingPtr<void>□□□□□□□□□□□□□□□□□□□□□□□□□□□□void*□□□
□□□□□□□□□□□□□□□void□□□□□□□□□□□□□□□□□□□□□policy□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□policy□□□□□□□

□□□□□□□□□□□□□□□□□□□□□CountingPtr□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□policy□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□

```
template<typename T,
        typename CounterPolicy = SimpleReferenceCount,
        typename ObjectPolicy = StandardObjectPolicy>
    class CountingPtr : private CounterPolicy, private
  ObjectPolicy {
        private:
          // typedef□□□□□□:
          typedef CounterPolicy CP;
          typedef ObjectPolicy OP;
          …
        public:
          // □□□□□□□□□□,□□□□□□□□□
    //□□□□□□□□cp□□□□□□□□□object_pointed_to□
        template<typename T2, typename CP2, typename
    OP2>
```

```
        class CountingPtr;
        friend
        template <typename S>    // S[illegible] void [illegible]T[illegible]
        CountingPtr(CountingPtr<S, OP, CP> const& cp)
          : OP((OP const&)cp),
            CP((CP const&)cp),
            object_pointed_to(cp.object_pointed_to) {
              if (cp.object_pointed_to != NULL) {
                CP::increment(cp.object_pointed_to);
              }
        }
      };
```

[illegible]

[illegible]bool[illegible]CountingPtr[illegible]

```
    template<typename T,
             typename CounterPolicy = SimpleReferenceCount,
             typename ObjectPolicy = StandardObjectPolicy>
    class CountingPtr : private CounterPolicy, private ObjectPolicy {
      ...
      public:
        operator bool() const {
          return this->object_pointed_to != (T*)0;
        }
    };
```

[illegible]CountingPtr[illegible]CountingPtr[illegible]

[illegible]bool[illegible]

if (cp) ...

[illegible]

while (!cp) ...

[illegible] void* [illegible]void*[illegible]bool [illegible] [8] [illegible]void*[illegible]

[illegible] bool [illegible]delete[illegible]CountingPtr[illegible]

```
template<typename T,
        typename CounterPolicy = SimpleReferenceCount,
        typename ObjectPolicy = StandardObjectPolicy>
class CountingPtr : private CounterPolicy, private ObjectPolicy {
  ...
  private:
    class BoolConversionSupport {
      int dummy;
    };
  public:
    operator BoolConversionSupport::*() const {
      return this->object_pointed_to
```

```
                ? &BoolConversionSupport::dummy
                : 0;
        }
    ...
};
```

[illegible]CountingPtr[illegible]

20.2.9 [illegible]

[illegible]= =[illegible]!=[illegible]<[illegible]<=[illegible]

[illegible]CountingPtrs[illegible]CountingPtr[illegible]

[illegible]= =[illegible]!=[illegible]

```
template<typename T,
         typename CounterPolicy = SimpleReferenceCount,
         typename ObjectPolicy = StandardObjectPolicy>
class CountingPtr : private CounterPolicy, private ObjectPolicy {
  ...
  public:
    friend bool operator==(CountingPtr<T,CP,OP> const& cp,
                           T const* p) {
        return cp == p;
```

```
        }
        friend bool operator==(T const* p,
                               CountingPtr<T,CP,OP> const& cp) {
            return p == cp;
        }
};
template <typename T1, typename T2,
          typename CP, typename OP>
inline
bool operator== (CountingPtr<T1,CP,OP> const& cp1,
                 CountingPtr<T2,CP,OP> const& cp2)
{
    return cp1.operator->() == cp2.operator->();
}
```

[illegible] CountingPtr [illegible] operator-> [illegible]

[illegible] 0 [illegible]

20.3 [illegible]

[illegible]

[MeyersMoreEffective] [illegible] [AlexandrescuDesign] [illegible] policy [illegible]

C++[illegible] auto_ptr[illegible] Holder/Trule [illegible]auto_ptr [illegible] C++[illegible] Holder/Trule[illegible]aotu_ptr[illegible]2[illegible] [9] [illegible]

[illegible]C++[illegible]C++[illegible]C++[illegible]

Boost [illegible] [BoostSmartPtr][illegible]

[illegible]21[illegible] tuple

[illegible] C/C++[illegible]C/C++[illegible]tuple[illegible]duo [10] [illegible]——duo[illegible] std::pair [illegible]duo[illegible]trio[illegible]quartet[illegible] [11] [illegible]

21.1 duo

duo[illegible]std::pair[illegible]pair[illegible]pair[illegible]duo[illegible]

```
template <typename T1, typename T2>
struct Duo {
    T1 v1;    // [illegible]1[illegible]
    T2 v2;    // [illegible]2[illegible]
```

```
};
```

[illegible]duo[illegible]

```
Duo<bool,X> result = foo();
if (result.v1) {
        // [illegible]result.v2
      …
}
```

[illegible]

[illegible]duo[illegible]2[illegible]duo[illegible]

```
template <typename T1, typename T2>
class Duo {
   public:
      T1 v1;    // [illegible]1[illegible]
      T2 v2;    // [illegible]2[illegible]
      // [illegible]
      Duo() : v1(), v2() {
      }
      Duo (T1 const& a, T2 const& b)
         : v1(a), v2(b) {
      }
};
```

[illegible]0[illegible]5.5[illegible]

[illegible]

```
template <typename T1, typename T2>
```

```
inline
Duo<T1,T2> make_duo (T1 const& a, T2 const& b)
{
   return Duo<T1,T2>(a,b);
}
```

[illegible]duo[illegible]

```
Duo<bool,int> result;
result.v1 = true;
result.v2 = 42;
return result;
```

[illegible]4[illegible]

```
return make_duo(true,42);
```

[illegible]C++[illegible]

```
return Duo<bool,int>(true,42);
```

[illegible] duo [illegible] duo [illegible]adapter template[illegible]

```
template <typename T1, typename T2>
class Duo {
   public:
      typedef T1 Type1;   // [illegible]1[illegible]
      typedef T2 Type2;   // [illegible]2[illegible]
      enum { N = 2 };    // [illegible]
      T1 v1;              // [illegible]1[illegible]
      T2 v2;              //[illegible]1[illegible]
      // [illegible]
      Duo() : v1(), v2() {
      }
```

```
        Duo (T1 const& a, T2 const& b)
            : v1(a), v2(b) {
        }
};
```

□□□□□□□□□□□□duo□□□□□std::pair□□□□□□□□□□□□□□

• □□□□□□□□□□□

• □□□□□□□□□N□□□□□□□□□□

• □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

• □□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
// tuples/duo1.hpp
#ifndef DUO_HPP
#define DUO_HPP
template <typename T1, typename T2>
class Duo {
    public:
        typedef T1 Type1;   // □1□□□□□
        typedef T2 Type2;   // □2□□□□□
        enum { N = 2 };    // □□□□
    private:
        T1 value1;            // □1□□□□
        T2 value2;            // □2□□□□
    public:
        //□□□□
        Duo() : value1(), value2() {
        }
        Duo (T1 const & a, T2 const & b)
```

```
    : value1(a), value2(b) {
  }
  // [illegible]
  template <typename U1, typename U2>
  Duo (Duo<U1,U2> const & d)
    : value1(d.v1()), value2(d.v2()) {
  }
  // [illegible]
  template <typename U1, typename U2>
  Duo<T1, T2>& operator = (Duo<U1,U2> const & d)
{
    value1 = d.value1;
    value2 = d.value2;
    return *this;
  }
  // [illegible]
  T1& v1() {
    return value1;
  }
  T1 const& v1() const {
    return value1;
  }
  T2& v2() {
    return value2;
  }
  T2 const& v2() const {
    return value2;
```

```
        }
    };
    // [illegible] ([illegible]):
    template <typename T1, typename T2,
        typename U1, typename U2>
    inline
    bool operator == (Duo<T1,T2> const& d1, Duo<U1,U2>
const& d2)
    {
      return d1.v1()==d2.v1() && d1.v2()==d2.v2();
    }
    template <typename T1, typename T2,
        typename U1, typename U2>
    inline
    bool operator != (Duo<T1,T2> const& d1, Duo<U1,U2>
const& d2)
    {
      return !(d1==d2);
    }
    // [illegible]
    template <typename T1, typename T2>
    inline
    Duo<T1,T2> make_duo (T1 const & a, T2 const & b)
    {
      return Duo<T1,T2>(a,b);
    }
    #endif // DUO_HPP
```

[illegible]

• [illegible]private[illegible]

• [illegible]

```
template <typename T1, typename T2>
class Duo {
  ...
  Duo[illegible] [illegible]value1[illegible] [illegible]value2[illegible] [illegible]{
  }
  ...
}
```

• [illegible]

• [illegible]= =[illegible]!=[illegible]duo[illegible]

[illegible]duo[illegible]duo[illegible]

```
// tuples/duo1.cpp
#include "duo1.hpp"
Duo<float,int> foo ()
{
    return make_duo(42,42);
}
int main()
{
  if (foo() == make_duo(42,42.0)) {
    ...
  }
}
```

□□□□□□foo()□□□□□□□□make_duo()□□□□□□□Duo<int,int>□□foo()□□□□□□□Duo<float,int>□□□□□□□□□□□□□□□main□□□□□□□□foo()□□□□□ make_duo(42,42.0) □□ Duo<int,double>□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□3□□□trio□□□duo□□□□□□□□□□□□

21.2 □□□duo

□□□□□□□□□□

Duo<int, Duo<char, Duo<bool, double> > > q4;

q4□□□□□□□□□□□duo□□□□□□□□□duo□□□□□□□□□□□2□□□□□□□□□□□duo□□□□□□□□□□□□□□□□1□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□2□□□□□□□□□□duo□□□□□□□2□□□□□□□□□□□duo□

21.2.1 □□□□

```
// tuples/duo2.hpp
template <typename A, typename B, typename C>
class Duo<A, Duo<B,C> > {
  public:
    typedef A          T1;              // □1□□□□□
    typedef Duo<B,C> T2;              // □2□□□□□
    enum { N = Duo<B,C>::N + 1 }; // □□□□
  private:
    T1 value1;                          // □1□□□□
    T2 value2;                          // □2□□□□
  public:
    // □□□□□□□□□□□□□
```

```
        ...
    };
```

□□□□□□□□□□□□□□duo□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□duo□

```
// tuples/duo6.hpp
// □□□□□□□□□Duo<>□□□□□template <typename A>
struct Duo<A,void> {
  public:
    typedef A T1;       // □1□□□□□
    typedef void T2;   // □2□□□□□
    enum { N = 1 };    // □□□□
  private:
    T1 value1;          // □2□□□□
  public:
    // □□□□
    Duo() : value1() {
    }
    Duo (T1 const & a)
      : value1(a) {
    }
    // □□□□□
    T1& v1() {
      return value1;
    }
    T1 const& v1() const {return value1;
  }
  void v2() {
```

```
    }
    void v2() const {
    }
    ...
};
```

[illegible]v2()[illegible]

[illegible]

21.2.2 [illegible]

[illegible]trio[illegible]quartet[illegible]duo[illegible]q4[illegible]3[illegible]

```
q4.v2().v2().v1()
```

[illegible]duo[illegible]

[illegible]DuoT[illegible]tuples/duo3.hpp[illegible]duo[illegible]n[illegible]

```
// [illegible]duo[illegible]N[illegible] T[illegible]
template <int N, typename T>
class DuoT {
  public:
    typedef void ResultT;    // [illegible]void
};
```

[illegible] non-Duo[illegible] duo[illegible] void[illegible]duo[illegible]

```
// [illegible]duo[illegible]1[illegible]
template <typename A, typename B>
class DuoT <1, Duo<A,B> > {
```

```
    public:
        typedef A ResultT;
};
// □□□□duo□2□□□□□
template <typename A, typename B>
class DuoT<2, Duo<A,B> > {
    public:
        typedef B ResultT;
};
```

□□□□□□□□□□□□□□□□□□□□□□□□duo□□N□□□□□□□□□□□□□□□□□2□□□□□□□□□duo□□□N-1□□□□□□

```
// □□□□□duo□N□□□□□□□□
template <int N, typename A, typename B, typename C>
class DuoT<N, Duo<A, Duo<B,C> > > {
    public:
        typedef typename DuoT<N-1, Duo<B,C> >::ResultT ResultT;
};
```

□□□□□□□□duo□1□□□□□□□□□□□□□□□□□□□□□□

```
// □□□□□duo□1□□□□□
template <typename A, typename B, typename C>
class DuoT<1, Duo<A, Duo<B,C> > > {
    public:
        typedef A ResultT;
};
```

□□□□□duo□2□□□□□□□□□□□□□□□□□□□duo□□□□□□□□□□□□□□□
□□□□□□□

```
// □□□□□duo□□2□□□□□□□
template<typename A, typename B, typename C>
class DuoT<2, Duo<A, Duo<B, C> > > {
  public:
    typedef B ResultT;
};
```

□□□□□□□□□□□□□□DuoT□□□□□□□□□□□□□□□□□□□□□IfThenElse□□□□15.2.4□□□□□□□□□□□□□

21.2.3 □□□

□□□□□□duo□□□□□□□□□□□N□□□□□□N□□□□□□□□□□□□□N□□□□□□□□□□□□□□□□□N□□□□□□□□□□□□□val<N>(duo)□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□DuoValue□□N□□□□□□□val()□□□□□□□□□□□□

```
// tuples/duo5.hpp
#include "typeop.hpp"
// □□□□duo□□N□□
template <int N, typename A, typename B>
inline
typename TypeOp<typename DuoT<N, Duo<A, B> >::ResultT>::RefT
val(Duo<A, B>& d)
{
  return DuoValue<N, Duo<A, B> >::get(d);
}
```

```
// □□□□duo□□N□□
template <int N, typename A, typename B>
inline
typename TypeOp<typename DuoT<N, Duo<A, B> >::ResultT>::RefConstT
val(Duo<A, B> const& d)
{
   return DuoValue<N, Duo<A, B> >::get(d);
}
```

□□□□□□□□□□□□□□□DuoT□□□□val□□□□□□□□□□□□□□15.2.3□□□□TypeOp□□□□□□□□□□□□□□□□□

□□□DuoValue□□□□□□□□□□□□□□DuoT□□□□□□□□□□□□□□□□

```
// tuples/duo4.hpp
#include "typeop.hpp"
// □□□□□□□□   (duo) T □□N□□
template <int N, typename T>
class DuoValue {
   public:
      static void get(T&) {          // □□□□□□□□□□□
      }
      static void get(T const&) {
      }
};
// □□□□duo□□1□□□□□
template <typename A, typename B>
class DuoValue<1, Duo<A, B> > {
```

```
    public:
      static A& get(Duo<A, B> &d) {
        return d.v1();
    }
    static A const& get(Duo<A, B> const &d) {
      return d.v1();
    }
  };
  // □□□□duo□2□□□□□
  template <typename A, typename B>
  class DuoValue<2, Duo<A, B> > {
    public:
      static B& get(Duo<A, B> &d) {
        return d.v2();
      }
      static B const& get(Duo<A, B> const &d) {
        return d.v2();
      }
  };
  // □□□□□duo□□N□□□□□
  template <int N, typename A, typename B, typename
C>
  struct DuoValue<N, Duo<A, Duo<B,C> > > {
    static
    typename TypeOp<typename DuoT<N-1, Duo<B,C>
 >::ResultT>::RefT
    get(Duo<A, Duo<B,C> > &d) {
```

```
        return DuoValue<N-1, Duo<B,C> >::get(d.v2());
    }
    static    typename    TypeOp<typename    DuoT<N-1,
Duo<B,C>
                               >::ResultT>::RefConstT
    get(Duo<A, Duo<B,C> > const &d) {
        return DuoValue<N-1, Duo<B,C> >::get(d.v2());
    }
};
// □□□□□duo□□1□□□□□
template <typename A, typename B, typename C>
class DuoValue<1, Duo<A, Duo<B,C> > > {
    public:
        static A& get(Duo<A, Duo<B,C> > &d) {
            return d.v1();
        }
        static A const& get(Duo<A, Duo<B,C> > const &d) {
            return d.v1();
        }
};
// □□□□□duo□□2□□□□□
template <typename A, typename B, typename C>
class DuoValue<2, Duo<A, Duo<B,C> > > {
    public:
        static B& get(Duo<A, Duo<B,C> > &d) {
            return d.v2().v1();
        }
```

```
        static B const& get(Duo<A, Duo<B,C> > const &d) {
            return d.v2().v1();
        }
};
```

□DuoT□□□□□□□□□□DuoValue□□□□□□□□get□□□□void□□□□□□□□□□□void□□□□□□□□□□DuoValue□□□□□nonduo□□□□□□□□□□N□□□□□□nonduo□□□□□N□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
//□□□□□□□□ (duo) T □□N□□
template <int N, typename T>
class DuoValue {
    public:
        static void get(T&) {          // □□□□□□□□□□□
        }
        static void get(T const&) {
        }
};
```

□DuoT□□□□□□□□□□□duo:

```
// □□□□duo□□1□□□□□
template <typename A, typename B>
class DuoValue<1, Duo<A, B> > {
    public:
        static A& get(Duo<A, B> &d) {
            return d.v1();
        }
        static A const& get(Duo<A, B> const &d) {
            return d.v1();
        }
```

```
};
...
□□□□□□□□duo□□□□□□DuoT□□□□
template <int N, typename A, typename B, typename
C>
class DuoValue<N, Duo<A, Duo<B,C> > > {
  public:
  static
  typename TypeOp<typename DuoT<N-1, Duo<B,C>
 >::ResultT>::RefT
  get(Duo<A, Duo<B,C> > &d) {
    return DuoValue<N-1, Duo<B,C> >::get(d.v2());
  }
  static typename TypeOp<typename DuoT<N-1,
 Duo<B,C>
                    >::ResultT>::RefConstT
  get(Duo<A, Duo<B,C> > const &d) {
    return DuoValue<N-1, Duo<B,C> >::get(d.v2());
  }
};
// □□□□□duo□□1□□□□□
template <typename A, typename B, typename C>
class DuoValue<1, Duo<A, Duo<B,C> > > {
  public:
    static A& get(Duo<A, Duo<B,C> > &d) {
      return d.v1();
    }
```

```
        static A const& get(Duo<A, Duo<B,C> > const &d) {
            return d.v1();
        }
};
// □□□□□duo□□2□□□□□
template <typename A, typename B, typename C>
class DuoValue<2, Duo<A, Duo<B,C> > > {
    public:
        static B& get(Duo<A, Duo<B,C> > &d) {
            return d.v2().v1();
        }
        static B const& get(Duo<A, Duo<B,C> > const &d) {
            return d.v2().v1();
        }
};
```

□□□□□□□□□□□□□duo□

```
// tuples/duo5.cpp
#include "duo1.hpp"
#include "duo2.hpp"
#include "duo3.hpp"
#include "duo4.hpp"
#include "duo5.hpp"
#include <iostream>
int main()
{
        // □□□□□□□□□□duo
        Duo<bool,int> d;
```

```
        std::cout << d.v1() << std::endl;
        std::cout << val<1>(d) << std::endl;
        // [illegible] triple
        Duo<bool,Duo<int,float> > t;
        val<1>(t) = true;
        val<2>(t) = 42;
        val<3>(t) = 0.2;
        std::cout << val<1>(t) << std::endl;
        std::cout << val<2>(t) << std::endl;
        std::cout << val<3>(t) << std::endl;
}
```

[illegible]

```
val<3>(t)
```

[illegible]

```
t.v2().v2()
```

[illegible]val[illegible]

[illegible]duo[illegible]

21.3 tuple[illegible]

[illegible]duo[illegible]metaprogramming[illegible]tuple[illegible]duo[illegible]duo[illegible]tuple[illegible]5[illegible]tuples/tuple1.hpp[illegible]

□□□ tuple □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□null□□□□□□□□□□□□NullT□□□□□□□null□□□□□□□NullT□□□□void□□□□□□□□□□□□NullT□□□□□void□□□□□□□□□

```
// □□□□□□□□□□
class NullT {
};
```

□□□□□□tuple□□□□□□□□□□duo□□□duo□□□□□□□NullT□□□□□

```
// □□□□□Tuple<>□□□“□□□□□NullT□□□Tuple<>”
template<typename P1,
         typename P2 = NullT,
         typename P3 = NullT,
         typename P4 = NullT,
         typename P5 = NullT>
class Tuple
  : public Duo<P1, typename
 Tuple<P2,P3,P4,P5,NullT>::BaseT> {
    public:
      typedef Duo<P1, typename
  Tuple<P2,P3,P4,P5,NullT>::BaseT>
         BaseT;
      // □□□:
      Tuple() {}
      Tuple(TypeOp<P1>::RefConstT a1,
         TypeOp<P2>::RefConstT a2,
         TypeOp<P3>::RefConstT a3 = NullT(),
         TypeOp<P4>::RefConstT a4 = NullT(),
```

```
        TypeOp<P5>::RefConstT a5 = NullT())
      : BaseT(a1, Tuple<P2,P3,P4,P5,NullT>
  (a2,a3,a4,a5)) {
    }
};
```

□□□1□□□□□□□□□□□□□□□□□duo□□□duo□□□□□□T1□T2□□□□□□□Pn□□□Tn□□□□□□□□□□ [12] □

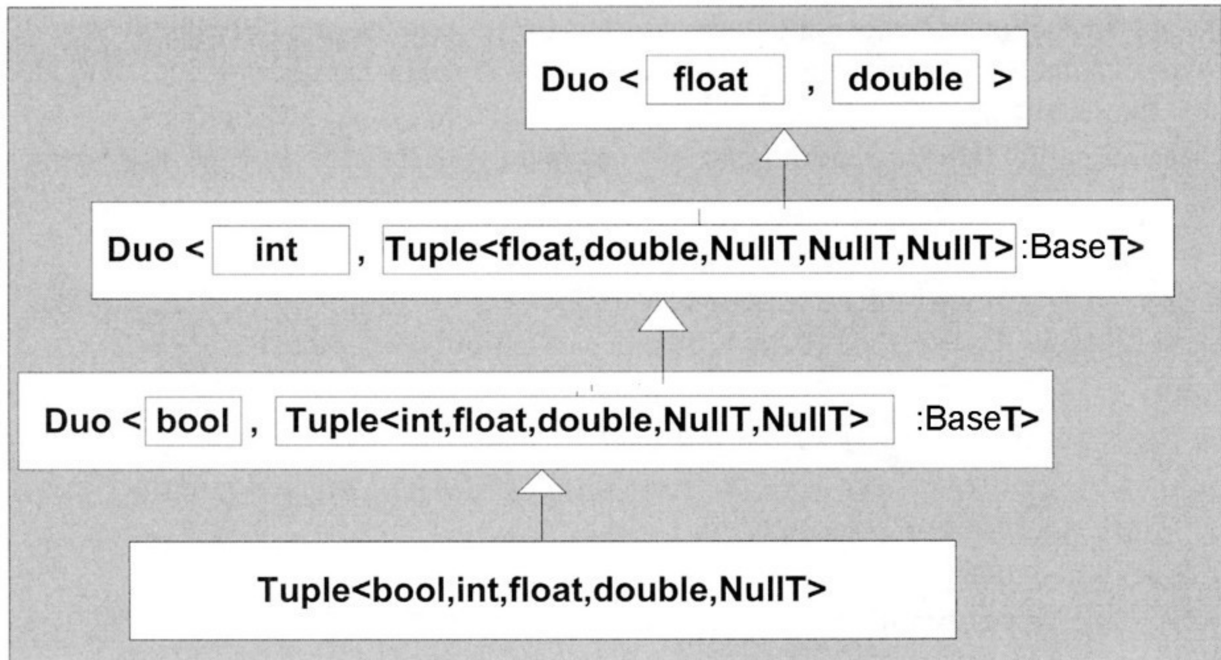
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□duo□

```
// □□□□□□□□□
template <typename P1, typename P2>
class Tuple<P1,P2,NullT,NullT,NullT> : public
Duo<P1,P2> {
  public:
    typedef Duo<P1,P2> BaseT;
    Tuple() {}
    Tuple(TypeOp<P1>::RefConstT a1,
      TypeOp<P2>::RefConstT a2,
      TypeOp<NullT>::RefConstT = NullT(),
      TypeOp<NullT>::RefConstT = NullT(),
      TypeOp<NullT>::RefConstT = NullT())
    : BaseT(a1, a2) {
  }
};
```

□□□□□□□□□□□□□

```
Tuple<bool,int,float,double> t4(true,42,13,1.95583);
```

□□□□□□□□□□□21.1□□□



□21.1 Tuple<bool,int,float,double>□□□

□□□□□□□□□□tuple□□□singleton□□□□□□□□□□□□□

```
// □□singletons□□□
template <typename P1>
class Tuple<P1,NullT,NullT,NullT,NullT> : public Duo<P1,void> {
  public:
    typedef Duo<P1,void> BaseT;
    Tuple() {}
    Tuple(TypeOp<P1>::RefConstT a1,
      TypeOp<NullT>::RefConstT = NullT(),
      TypeOp<NullT>::RefConstT = NullT(),
      TypeOp<NullT>::RefConstT = NullT(),
      TypeOp<NullT>::RefConstT = NullT())
    : BaseT(a1) {
  }
```

```
};
```

□□□□□□□□□□□21.1□□make_duo()□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ tuple□□□□□□□□□□□□□□make_duo()□□□□□□□□□□□□□□□□ [13] □□□

```
// □□□□□□□□□□
template <typename T1>
inline
Tuple<T1> make_tuple(T1 const &a1)
{
   return Tuple<T1>(a1);
}
// □□□□□□□□□□
template <typename T1, typename T2>
inline
Tuple<T1,T2> make_tuple(T1 const &a1, T2 const &a2)
{
   return Tuple<T1,T2>(a1,a2);
}
// □□3□□□□□□□
template <typename T1, typename T2, typename T3>
inline
Tuple<T1,T2,T3> make_tuple(T1 const &a1, T2 const
&a2,
                              T3 const &a3)
{
   return Tuple<T1,T2,T3>(a1,a2,a3);
```

```
}
// □□4□□□□□□□□
template <typename T1, typename T2, typename T3,
typename T4>
inline
Tuple<T1,T2,T3,T4> make_tuple(T1 const &a1, T2 const
&a2,
                         T3 const &a3, T4 const &a4)
{
   return Tuple<T1,T2,T3,T4>(a1,a2,a3,a4);
}
// □□5□□□□□□□□
template <typename T1, typename T2, typename T3,
typename T4, typename T5>
inline
Tuple<T1,T2,T3,T4,T5> make_tuple(T1 const &a1, T2
const &a2,
                         T3 const &a3, T4 const &a4,
                         T5 const &a5)
{
   return Tuple<T1,T2,T3,T4,T5>(a1,a2,a3,a4,a5);
}
```

□□□□□□□□□□□□tuple□

```
// tuples/tuple1.cpp
#include "tuple1.hpp"
#include <iostream>
int main()
```

```
{
    // □□□□□□□□1□□□□tuple
    Tuple<int> t1;
    val<1>(t1) += 42;
    std::cout << t1.v1() << std::endl;
    // □□□□□duo
    Tuple<bool,int> t2;
    std::cout << val<1>(t2) << ", ";
    std::cout << t2.v1() << std::endl;
    // □□□□□triple
    Tuple<bool,int,double> t3;
    val<1>(t3) = true;
    val<2>(t3) = 42;
    val<3>(t3) = 0.2;
    std::cout << val<1>(t3) << ", ";
    std::cout << val<2>(t3) << ", ";
    std::cout << val<3>(t3) << std::endl;
    t3 = make_tuple(false, 23, 13.13);
    std::cout << val<1>(t3) << ", ";
    std::cout << val<2>(t3) << ", ";
    std::cout << val<3>(t3) << std::endl;
    // □□□□□ quadruple
    Tuple<bool,int,float,double> t4(true,42,13,1.95583);
    std::cout << val<4>(t4) << std::endl;
    std::cout << t4.v2().v2().v2() << std::endl;
}
```

□□tuple□□

```
Tuple<bool,int,float> t3;
t3 = make_tuple(false, 23, 13.13); // □□: 13.13 □double□□
```

21.4 □□□□

□□□□□□□□□□□□□□□□□□□□tuple□□□□□□□□□□□□□tuple□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□pair□□□□□□□□□□□□duo□□□□□□□Andrei Alexandrescu□[AlexandrescuDesign]□□□□□□□□□□□□□□□□tuple□□□□□□list of type□□tuple□□□□□list of field□□□□□□□□□□□type list□□□□□□□□□□typelist□□□□□□□□□□□□□□□□□□□□□□tuple□□□□

13.3□□□□template list parameter□□□□□□□□□□□□□□□□□□tuple□□□□

□22□ □□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□C□□□□□□□□3□□□ C □□□□□□□□□□□□□□□□□□C++□□□□□□□□□□□□□class□□□□□□□□□□□□□□□□□□□□□□□reference to function□□□

[illegible]C++[illegible]“[illegible]”[illegible]

[illegible]C++[illegible]class[illegible]“[illegible]class[illegible]”[illegible]“[illegible]”[illegible]“[illegible]class[illegible]”[illegible]“[illegible]”[illegible]

[illegible]

22.1 [illegible]

[illegible]C[illegible]C++[illegible]C[illegible]C++[illegible]mangled name[illegible]decorated name[illegible]

f();

[illegible] f [illegible] f[illegible]

[illegible]——[illegible] [14] [illegible]

[illegible]

```
void foo (void (*pf)())
{
   pf();   // [illegible] pf [illegible]
}
```

[illegible]pf[illegible]foo()[illegible]pf[illegible]pf[illegible]

[illegible]

```
int f1(int const & r)
{
   return ++(int&)r;    // [illegible]
}
int f2(int const & r)
{
   return r;
}
int f3()
{
```

```
    return 42;
}
int foo()
{
    int param = 0;
    int answer = 0;
    answer = f1(param);
    f2(param);
    f3();
    return answer + param;
}
```

[illegible]f1[illegible]const int[illegible]const[illegible]C++[illegible]const[illegible]C++[illegible]f1[illegible]const[illegible]

[illegible]f2()[illegible]param[illegible]param[illegible]f3()[illegible]param[illegible]f1()[illegible]f2()[illegible]param[illegible]f3()[illegible]param[illegible]

[illegible]C++[illegible]f1()[illegible]r[illegible]foo()[illegible]param[illegible]

[illegible]C[illegible]C++[illegible] [15] [illegible]bug[illegible]

[illegible]f1()[illegible]f2()[illegible]f3()[illegible]foo()[illegible]

```
int foo'()
{
   int param = 0;
   int answer = 0;
   answer = ++(int&)param;
   return answer + param;
}
```

[illegible]

```
int foo''()
{
   return 2;
}
```

[illegible]

[illegible]

22.2 [illegible]

[illegible]foo()[illegible]

```
extern "C++" void foo() throw()
{
}
```

[illegible]C++[illegible]C++[illegible] [16] [illegible]C[illegible]C++[illegible]C++[illegible]C[illegible]C++[illegible]C [illegible] C++[illegible]C++[illegible]

[illegible]foo[illegible]foo()[illegible]foo[illegible]ia[illegible]

```
int ia[10];
```

ia[illegible]1[illegible] decay[illegible]C++[illegible]

```
// functors/funcptr.cpp
#include <iostream>
#include <typeinfo>
void foo()
{
   std::cout << "foo() called" << std::endl;
}
typedef void FooT(); // FooT [illegible],
            // [illegible] foo()[illegible]
int main()
{
```

```
    foo();               // □□□□
    // □□ foo □ FooT □□□
    std::cout << "Types of foo:   " << typeid(foo).name()
      << '\n';
    std::cout << "Types of FooT: " << typeid(FooT).name()
      << '\n';
    FooT* pf = foo;   // □□□□(decay)
    pf();                // □□□□□□□□
    (*pf)();             // □□□pf()
    // □□□pf□□□
    std::cout << "Types of pf:    " << typeid(pf).name()
      << '\n';
    FooT& rf = foo;   // □□□□□
    rf();                // □□□□□□□□
    // □□rf□□□
    std::cout << "Types of rf:    " << typeid(rf).name()
      << '\n';
}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□typeid□□□□□□□□□□□□□std::type□□□std::type□name()□□□□□□□□□□□□□□□□5.6□□□□□□□□□typeid□□□□□□□□□□□□□□□□□decay□

□□□□C++□□□□□□□

```
foo() called
Types of foo:   void ()
Types of FooT: void ()
foo() called
```

foo() called

Types of pf: FooT *

foo() called

Types of rf: void ()

[illegible] name()[illegible] typedef[illegible]5[illegible]FooT*[illegible]void(*) [illegible]

[illegible]foo[illegible]non-const[illegible]

[illegible]pf[illegible]rf[illegible]——[illegible]

22.3 [illegible]

[illegible]C++[illegible] p->mf()[illegible]p[illegible]mf()[illegible]this[illegible]mf()[illegible]

[illegible]mf()[illegible]p[illegible]

```
class B1 {
  private:
    int b1;
  public:
    void mf1();
};
```

```
void B1::mf1()
{
   std::cout << "b1="<<b1<<std::endl;
}
```

□□□□□□□□□mf1()□□□□□□B1□□□□□□□□□□□this□□□□□□□□□B1□□□□

□□□□□□□□□□□□□□□□□□□

```
class B2 {
   private:
      int b2;
   public:
      void mf2();
};
void B2::mf2()
{
      std::cout << "b2="<<b2<<std::endl;
}
```

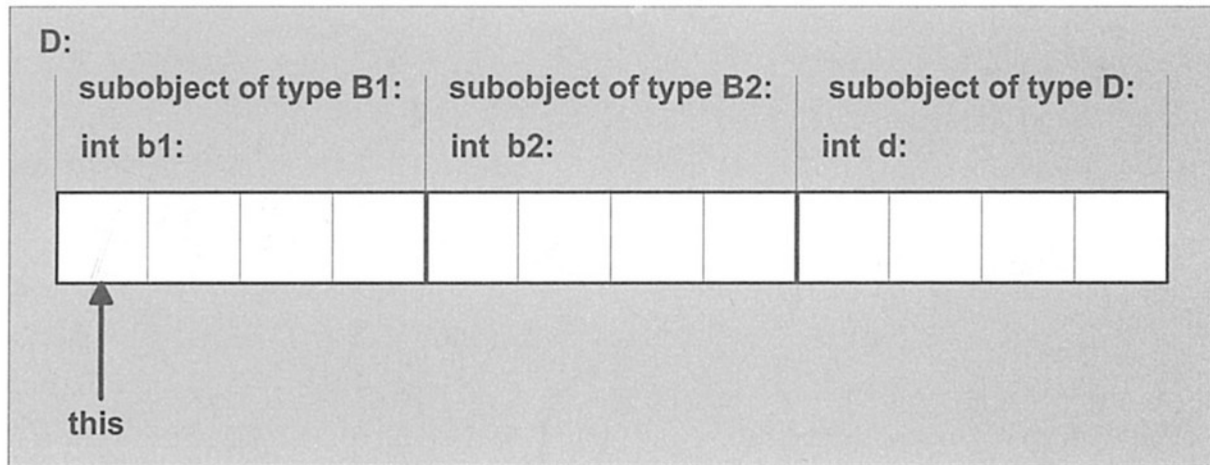
□□□□□□□□mf2()□□□□□□□□this□□□□□□B2□□□□□□□

□□□□□□□□□□□□□□B1□B2□□□□

```
class D: public B1, public B2 {
   private:
      int d;
};
```

□□□□□□□□□□□D□□□□□□□□B1□□□□□□□□□□□□□B2□□□□□□□□□□□□D□□□□□□□□□□□□D□□□□□□□□□□B1□□□□□□□□B2□□□□□□□□□□□□□□□□□32□□□□□□D□□□□□□□□□□□□□□□□22.1□□□□□□□□□□int□□□□4□□□□□□□□□□b1□□□□this□□□□□□b2□□□□this□□□□□4□□□□□□□d□□□

□this□□□□8□□□□B1□B2□□□□□□□B1□□□□□b1□□D□□□□□□□□□□□
□□this□□□□□B2□□□□□□b2□□□□□



□22.1 □□D□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□

```
int main()
{
   D obj;
   obj.mf1();
   obj.mf2();
}
```

□□ obj.mf2()□□□obj □ B2 □□□□□□□□□□□□□□□□□□□□□□□□□□mf2()□□□□□□□□□□□□□22.1□□□□□□□□□□□□□□□□□obj□□□□□4□□□□□□□□□□□□□□□□C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□mf1()□□□□□□□□□□□□□□□□□□obj□□□□□□obj□□□B1□□□□□□□

□□main()□□□

```
void call_memfun (D obj, void (D::*pmf) ())
{
   (obj.*pmf)();
```

```
}
int main()
{
   D obj;
   call_memfun(obj, &D::mf1);
   call_memfun(obj, &D::mf2);
}
```

□□□C++□□□□□□□□□□□□□□□□□□call_memfun()□□□main()□□□□□□□□□□□□

□□□this□□□

```
void (D::*pmf_a) ()   = &D::mf2;                    // □□□□□
+4□□□
void (B2::*pmf_b)() = (void (B2::*)())pmf_a;   // □□□□□□□
                                   // □□□□□0
```

□□3-□□□□□□□□□3□□

1.□□□□□□□□□□□□□□□□□□□□□□NULL□

2.□□this□□□□□□

3.□□□□□□□

□□Stan Lippman□Inside C++ Object Model□□[LippmanObjMod]□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□this □□□□□□□□□□□□this□□

[illegible]2[illegible]pmf[illegible]obj[illegible] .* [illegible]->* [illegible]

[illegible]obj.*pmf[illegible](...) // [illegible]obj[illegible]pmf[illegible]

[illegible]ptr->*pmf[illegible](...) // [illegible]ptr[illegible]pmf[illegible]

[illegible]

(*ptr)()

[illegible]

ptr()

[illegible] [17] [illegible]

22.4 class[illegible]

[illegible]C++[illegible]class[illegible]

22.4.1 class[illegible]1[illegible]

[illegible]class[illegible]

```
// functors/functor1.cpp
#include <iostream>
// [illegible]
class ConstantIntFunctor {
private:
   int value;    //“[illegible]”[illegible]
public:
   // [illegible]
   ConstantIntFunctor (int c) : value(c) {
```

```
    }
    //“□□□□”
    int operator() () const {
      return value;
    }
  };
  // □□□□“□□□□”□□□□□□
  void client (ConstantIntFunctor const& cif)
  {
    std::cout << "calling back functor yields " << cif() <<
'\n';
  }
  int main()
  {
    ConstantIntFunctor seven(7);
    ConstantIntFunctor fortytwo(42);
    client(seven);
    client(fortytwo);
  }
```

ConstantIntFunctor□□□class□□

```
ConstantIntFunctor seven(7);   // □□□□□□□□
□□□□□□
seven();                       // □□□□□□□ operator ()
```

□□□□□□ seven □ operator()□□□□□□□□ seven()□□□□□□□□□□□□□seven□fortytwo□client()□□□cif□□□□□□□□□□□□□□□□□□□□□□□□

[illegible]class [illegible]class[illegible]seven[illegible]fortytwo[illegible]

22.4.2 class[illegible]

[illegible]class[illegible]class[illegible]

[illegible]

C++[illegible]set[illegible]map[illegible]Person[illegible]set[illegible]set[illegible]set[illegible]

```
#include <set>
class Person {
  …
};
class PersonSortCriterion {
  public:
    bool operator() (Person const& p1, Person const& p2)
const {
      // [illegible] p1 [illegible] ''[illegible]'' p2
      …
    }
};
```

```
void foo()
{
   std::set<Person, std::less<Person> > c0, c1;
                        //□operator < □□□□□□□□□
   std::set<Person, std::greater<Person> > c2;
                        // □ operator > □□□□□□□□□
   std::set<Person, PersonSortCriterion> c3;
   …                                                        //□□□□□□□□□□□□□
□□
   c0 = c1;             // □□: □□□□□
   c1 = c2;             // □□: □□□□□
   …
   if (c1 == c3) {    // □□: □□□□□
      …
   }
}
```

□□set□□3□□□std::less□operator< □□□□□“□□□□”□□□□□□□□□□□□□□□□□std::less□□□□□□□□□□ [18] □

```
namespace std {
   template <typename T>
   class less {
      public:
         bool operator() (T const& x, T const& y) const {
            returnx<y;
         }
   };
```

```
}
```

[illegible]std::greater[illegible]

[illegible]3[illegible]set[illegible]set[illegible]

22.5 [illegible]

[illegible] set [illegible]

22.5.1 [illegible]

[illegible] class[illegible]

[illegible]class[illegible]

```
template <typename FO>
void my_sort (... )
{
    FO cmp;             // [illegible]
    ...
    if (cmp(x,y)) {   //[illegible]2[illegible]
        ...
    }
    ...
```

```
}
// [illegible]
my_sort<std::less<... > > (... );
```

[illegible] std::less<>[illegible]cmp[illegible]

22.5.2 [illegible]

[illegible]

[illegible]0[illegible]my_sort[illegible]

```
template <typename F>
void my_sort (... , F cmp)
{
   ...
   if (cmp(x,y)) {   // [illegible]
      ...
   }
   ...
}
// [illegible]
my_sort (... , std::less<... >());
```

[illegible]my_sort()[illegible]cmp[illegible]

[illegible]“[illegible]”[illegible]“[illegible]—[illegible]”[illegible]

[illegible]

[illegible]

```
bool my_criterion () (T const& x, T const& y);
// [illegible]
my_sort (… , my_criterion);
```

[illegible]

22.5.3 [illegible]

[illegible]——[illegible]class[illegible]

```
template <typename F>
void my_sort (… , F cmp = F())
{
  …
  if (cmp(x,y)) {   // [illegible]
    …
  }
  …
}
bool my_criterion () (T const& x, T const& y);
// [illegible]
my_sort<std::less<… > > (… );
// [illegible]
```

```
my_sort (… , std::less<… >());
// [illegible]
my_sort (… , my_criterion);
```

C++[illegible]

```
class RuntimeCmp {
  …
};
// [illegible]
// ([illegible])
set<int,RuntimeCmp> c1;
// [illegible]
set<int,RuntimeCmp> c2(RuntimeCmp(… ));
```

[illegible][JosuttisStdLib][illegible]178[illegible]197[illegible]

22.5.4 [illegible]

[illegible] 4.3 [illegible]8.3.3[illegible]class[illegible]class[illegible]

```
class MyCriterion {
  public:
    bool operator() (SomeType const&, SomeType const&) const;
};
template <MyCriterion F>     // ERROR: MyCriterion[illegible]class[illegible]
void my_sort (… );
```

[illegible]class[illegible]

```
class MyCriterion {
  public:
    virtual bool operator() (SomeType const&,
                             SomeType const&) const = 0;
};
class LessThan : public MyCriterion {
  public:
    virtual bool operator() (SomeType const&,
                             SomeType const&) const;
};
template<MyCriterion& F>
void sort (... );
LessThan order;
sort<order> (... );                    // [illegible]: [illegible]
sort<(MyCriterion&)order> (... );   // [illegible]
                             //[illegible]
```

[illegible]LessThan[illegible]MyCriterion[illegible]C++[illegible]C++[illegible]

[illegible]class[illegible]

22.5.5 [illegible]

[illegible]class[illegible]

[illegible]

class CriterionWrapper {

```
class CriterionWrapper {
  public:
    bool operator() (… ) {
      return wrapped_function(… );
    }
};
```

[illegible]wrapped_function(…)[illegible]class[illegible]

[illegible]class[illegible]

```
template<int (*FP)()>
class FunctionReturningIntWrapper {
  public:
    int operator() () {
      return FP();
    }
};
```

[illegible]

```
// functors/funcwrap.cpp
#include <vector>
#include <iostream>
#include <cstdlib>
// [illegible]
```

```
template<int (*FP)()>
class FunctionReturningIntWrapper {
  public:
    int operator() () {
      return FP();
    }
};
// □□□□□□□□□□
int random_int()
{
  return std::rand(); // □□□□□C□□
}
// □□□□□□□□□□□□□□□□□□□□□□□
template <typename FO>
void initialize (std::vector<int>& coll)
{
  FO fo; // □□□□□□
  for (std::vector<int>::size_type i=0; i<coll.size(); ++i)
{
    coll[i] = fo(); // □□□□□□□□□□□□
  }
}
int main()
{
  // □□□□10□□□□vector
  std::vector<int> v(10);
  // □□□□□□□□□□□□□vector□□
```

```
    initialize<FunctionReturningIntWrapper<random_int> >(v);
    // [illegible]vector[illegible]
    for (std::vector<int>::size_type i=0; i<v.size(); ++i) {
        std::cout << "coll[" << i << "]: " << v[i] << std::endl;
    }
}
```

[illegible]initialize()[illegible]

FunctionReturningIntWrapper<random_int>

[illegible]random_int[illegible]

FunctionReturningIntWrapper<random_int>

[illegible]initialize[illegible]

[illegible] C [illegible] FunctionReturningIntWrapper[illegible]

```
initialize<FunctionReturningIntWrapper<std::rand> >(v);
```

[illegible]std::rand()[illegible]C[illegible]C[illegible] [19] [illegible]typedef[illegible]

```
// [illegible]C[illegible]
extern "C" typedef int (*C_int_FP)();
// [illegible]
template<C_int_FP FP>
class FunctionReturningIntWrapper {
  public:
    int operator() () {
        return FP();
    }
```

```
};
```

[illegible] FunctionReturningIntWrapper [illegible] FP [illegible] C++ [illegible]

22.6 [illegible]

[illegible] 15 [illegible] [20] [illegible] n [illegible]

[illegible] 2 [illegible]

```
class SuperFunc {
  public:
    void operator() (int, char**);
};
```

[illegible] C++ [illegible] —— [illegible] typeof [illegible] sizeof [illegible] typeof [illegible] typeof [illegible] 13.8 [illegible]

[illegible]

22.6.1 [illegible]

[illegible] class [illegible] [21] [illegible]

• [illegible]NumParams[illegible]

• [illegible]typedef Param1T[illegible]Param2T[illegible]Param3T[illegible]

• [illegible]typedef ReturnT[illegible]

[illegible]PersonSortCriterion[illegible]

```
class PersonSortCriterion {
  public:
    enum { NumParams = 2 };
    typedef bool ReturnT;
    typedef Person const& Param1T;
    typedef Person const& Param2T;
    bool operator() (Person const& p1, Person const& p2)
const {
      // [illegible]p1 [illegible] ''[illegible]'' p2
      …
    }
};
```

[illegible]PersonSortCriterion[illegible]typedef[illegible]

[illegible]pure functor[illegible] [22] [illegible]

22.6.2 [illegible]

[illegible]3[illegible]8[illegible]Param8T[illegible]N[illegible]N[illegible]

```
template<typename FunctorType, int N>
class FunctorParam;
```

[illegible]1[illegible]20[illegible]20[illegible]N[illegible]typedef Type[illegible]

[illegible]N[illegible]FunctorParam<F, N>::Type[illegible] FunctorParam [illegible] 2 [illegible]FunctorParam<F, N>::Type[illegible]void[illegible]void[illegible]void[illegible]void[illegible]3[illegible]FunctorParam<F, N>::Type[illegible]class[illegible]

```
// functors/functorparam1.hpp
#include "ifthenelse.hpp"
template <typename F, int N>
class UsedFunctorParam;
template <typename F, int N>
class FunctorParam {
  private:
    class Unused {
      private:
        class Private {};
      public:
        typedef Private Type;
```

```
    };
  public:
    typedef typename IfThenElse<F::NumParams>=N,
                                UsedFunctorParam<F,N>,
                                Unused>::ResultT::Type
          Type;
};
template <typename F>
class UsedFunctorParam<F, 1> {
  public:
    typedef typename F::Param1T Type;
};
```

IfThenElse □□□□ 15.2.4 □□□□□□□□□□□□□□□□□□□□ UsedFunctorParam□□□□□□□□N□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□

```
// functors/functorparam2.hpp
#define FunctorParamSpec(N)                                  \
  template<typename F>                                     \
  class UsedFunctorParam<F, N> {                          \
    public:                                                \
      typedef typename F::Param##N##T Type;    \
  }
…
FunctorParamSpec(2);
FunctorParamSpec(3);
…
FunctorParamSpec(20);
```

```
#undef FunctorParamSpec
```

22.6.3 [illegible]

[illegible]typedef[illegible]2[illegible]2[illegible]C++[illegible] C [illegible]

[illegible]2[illegible]FunctionPtr[illegible] func_ptr[illegible]FuntionPtr[illegible]

```
template<typename RT, typename P1 = void, typename P2 = void>
class FunctionPtr;
```

[illegible]void[illegible]

[illegible]

```
//functors/functionptrt.hpp
// [illegible]:
template<typename RT, typename P1 = void,
                      typename P2 = void,
                      typename P3 = void>
class FunctionPtrT {
  public:
    enum { NumParams = 3 };
    typedef RT (*Type)(P1,P2,P3);
```

```
};
//[illegible]
template<typename RT, typename P1,
                    typename P2>
class FunctionPtrT<RT, P1, P2, void> {
  public:
    enum { NumParams = 2 };
    typedef RT (*Type)(P1,P2);
};
// [illegible]:
template<typename RT, typename P1>
class FunctionPtrT<RT, P1, void, void> {
  public:
    enum { NumParams = 1 };
    typedef RT (*Type)(P1);
};
// [illegible]0[illegible]:
template<typename RT>
class FunctionPtrT<RT, void, void, void> {
  public:
    enum { NumParams = 0 };
    typedef RT (*Type)();
};
```

[illegible]

[illegible] class [illegible] class[illegible]class[illegible]

[illegible]class[illegible]class[illegible]const[illegible]15[illegible]TypeT[illegible]IfThenElse[illegible]

```
// functors/forwardparam.hpp
#ifndef FORWARD_HPP
#define FORWARD_HPP
#include "ifthenelse.hpp"
#include "typet.hpp"
#include "typeop.hpp"
// [illegible]class[illegible]ForwardParamT<T>::Type [illegible]
// [illegible]ForwardParamT<T>::Type[illegible]
// [illegible]void[illegible]ForwardParamT<T>::Type[illegible]Unused[illegible]
template<typename T>
class ForwardParamT {
  public:
    typedef typename IfThenElse<TypeT<T>::IsClassT,
                        typename TypeOp<T>::RefConstT,
                        typename TypeOp<T>::ArgT
                       >::ResultT
      Type;
};
template<>
class ForwardParamT<void> {
  private:
    class Unused {};
  public:
    typedef Unused Type;
```

```
};
#endif // FORWARD_HPP
```

[illegible]15.3.1[illegible]RParam[illegible]void[illegible]void[illegible]

[illegible]FunctionPtr[illegible]FunctionPtr[illegible]3[illegible]

```
// functors/functionptr.hpp
#include "forwardparam.hpp"
#include "functionptrt.hpp"
template<typename RT, typename P1 = void,
                      typename P2 = void,
                      typename P3 = void>
class FunctionPtr {
  private:
    typedef typename FunctionPtrT<RT,P1,P2,P3>::Type
FuncPtr;
    // [illegible]
    FuncPtr fptr;
  public:
    // [illegible]:
    enum { NumParams =
FunctionPtrT<RT,P1,P2,P3>::NumParams };
    typedef RT ReturnT;
    typedef P1 Param1T;
    typedef P2 Param2T;
```

```
      typedef P3 Param3T;
      // □□□□:
      FunctionPtr(FuncPtr ptr)
        : fptr(ptr) {
      }
      //“□□□□”:
      RT operator()() {
        return fptr();
      }
      RT  operator()(typename  ForwardParamT<P1>::Type
  a1) {
      return fptr(a1);
    }
    RT   operator()(typename   ForwardParamT<P1>::Type
a1,
                typename  ForwardParamT<P2>::Type  a2)
           {
      return fptr(a1, a2);
    }
    RT   operator()(typename   ForwardParamT<P1>::Type
a1,
                typename ForwardParamT<P2>::Type a2,
                typename  ForwardParamT<P3>::Type  a3)
           {
      return fptr(a1, a2, a3);
    }
  };
```

```
// functors/funcptr.hpp
#include "functionptr.hpp"
template<typename RT> inline
FunctionPtr<RT> func_ptr (RT (*fp)())
{
   return FunctionPtr<RT>(fp);
}
template<typename RT, typename P1> inline
FunctionPtr<RT,P1> func_ptr (RT (*fp)(P1))
{
   return FunctionPtr<RT,P1>(fp);
}template<typename RT, typename P1, typename P2> inline
FunctionPtr<RT,P1,P2> func_ptr (RT (*fp)(P1,P2))
{
   return FunctionPtr<RT,P1,P2>(fp);
}
template<typename RT, typename P1, typename P2, typename P3> inline
FunctionPtr<RT,P1,P2,P3> func_ptr (RT (*fp)(P1,P2,P3))
{
   return FunctionPtr<RT,P1,P2,P3>(fp);
}
```

```
// functors/functordemo.cpp
```

```
#include <iostream>
#include <string>
#include <typeinfo>
#include "funcptr.hpp"
double seven()
{
   return 7.0;
}
std::string more()
{
   return std::string("more");
}
template <typename FunctorT>
void demo (FunctorT func)
{
   std::cout << "Functor returns type "
         <<   typeid(typename   FunctorT::ReturnT).name()
   << '\n'
         << "Functor returns value "
         << func() << '\n';
}
int main()
{
   demo(func_ptr(seven));
   demo(func_ptr(more));
}
```

22.7 □□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
// functors/math1.hpp
#include <cmath>
#include <cstdlib>
class Abs {
  public:
    double operator() (double v) const {
    // ''□□□□'':
    return std::abs(v);
  }
};
class Sine {
  public:
    // ''□□□□'':
    double operator() (double a) const {
      return std::sin(a);
    }
};
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□sin□□

```
class AbsSine {
  public:
    double operator() (double a) {
      return std::abs(std::sin(a));
    }
```

```
};
```

[illegible]concept[illegible]

22.7.1 [illegible]

[illegible]

```
// functors/compose1.hpp
template <typename FO1, typename FO2>
class Composer {
  private:
    FO1 fo1;   // [illegible]1[illegible]/[illegible]
    FO2 fo2;   // [illegible]2[illegible]/[illegible]
  public:
    // [illegible]Composer (FO1 f1, FO2 f2)
      : fo1(f1), fo2(f2) {
    }
    //"[illegible]": [illegible]
    double operator() (double v) {
    return fo2(fo1(v));
  }
};
```

[illegible]Composer<Abs, Sine>[illegible]sin (abs (x))[illegible]

```
// functors/compose1.cpp
#include <iostream>
```

```
#include "math1.hpp"
#include "compose1.hpp"
template<typename FO>
void print_values (FO fo)
{
   for (int i=-2; i<3; ++i) {
      std::cout << "f(" << i*0.1
               << ") = " << fo(i*0.1)
               << "\n";
   }
}
int main()
{
   // □□ sin(abs(0.5))
   std::cout  <<  Composer<Abs,Sine>(Abs(),Sine())(0.5)
<< "\n\n";
   // □□□□□□ abs()print_values(Abs());
   std::cout << '\n';
   // □□□□□□ sin()print_values(Sine());
   std::cout << '\n';
   // □□□□□□ sin(abs())print_values(Composer<Abs, Sine>
(Abs(), Sine()));
   std::cout << '\n';
   // □□□□□□ abs(sin())
   print_values(Composer<Sine, Abs>(Sine(), Abs()));
}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

[illegible]

Composer[illegible]

```
// functors/composeconv.hpp
template <typename FO1, typename FO2>
inline
Composer<FO1,FO2> compose (FO1 f1, FO2 f2) {
   return Composer<FO1,FO2> (f1, f2);
}
```

[illegible]

```
// functors/compose2.cpp
#include <iostream>
#include "math1.hpp"
#include "compose1.hpp"
#include "composeconv.hpp"
template<typename FO>
void print_values (FO fo)
{
   for (int i=-2; i<3; ++i) {
      std::cout << "f(" << i*0.1
                << ") = " << fo(i*0.1)
                << "\n";
   }
}
int main()
{
   // [illegible] sin(abs(-0.5)) [illegible]
   std::cout << compose(Abs(),Sine())(0.5) << "\n\n";
```

```
    // □□□□□□ abs()
    print_values(Abs());
    std::cout << '\n';
    // □□□□□□ sin()
    print_values(Sine());
    std::cout << '\n';
    // □□□□□□ sin(abs())
    print_values(compose(Abs(),Sine()));
    std::cout << '\n';
    // □□□□□□ abs(sin())
    print_values(compose(Sine(),Abs()));
}
```

□□□□□□□□□□□□□

```
Composer<Abs, Sine>(Abs(), Sine())
```

□□□□□□□□□

```
compose(Abs(), Sine())
```

□□□□□□□□□Composer□□□□□□□□□□□□□Composer□□□□□□□□□first□second□□ 16.2 □□□□□□□□□□□□□

```
// functors/compose3.hpp
template <typename FO1, typename FO2>
class Composer : private FO1, private FO2 {
  public:
    // □□□□: □□□□□□
    Composer(FO1 f1, FO2 f2)
```

```
        : FO1(f1), FO2(f2) {
    }
    // ''□□□□'': □□□□□□□□□
    double operator() (double v) {
        return FO2::operator()(FO1::operator()(v));
    }
};
```

□□□

```
//□□□□□□ sin(sin())
print_values(compose(Sine(),Sine())); // □□: □□□□□□□
```

□□□Composer□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
// functors/compose4.hpp
template <typename C, int N>
class BaseMem : public C {
public:
    BaseMem(C& c) : C(c) { }
    BaseMem(C const& c) : C(c) { }
};
template <typename FO1, typename FO2>
class Composer : private BaseMem<FO1,1>,
                 private BaseMem<FO2,2> {
public:
    // □□□□: □□□□□□□
    Composer(FO1 f1, FO2 f2)
        : BaseMem<FO1,1>(f1), BaseMem<FO2,2>(f2) {
```

```
    }
    // ''[illegible]'': [illegible]
    double operator() (double v) {
        return BaseMem<FO2,2>::operator()
            (BaseMem<FO1,1>::operator()(v));
    }
};
```

[illegible]

[illegible] Composer [illegible]

22.7.2 [illegible]

[illegible] Composer [illegible]double[illegible]double[illegible]int[illegible]bool[illegible]bool[illegible]double[illegible]typedef[illegible]

[illegible] 22.6.1 [illegible] 3 [illegible]

```
// functors/compose5.hpp
#include "forwardparam.hpp"
template <typename C, int N>
class BaseMem : public C {
  public:
    BaseMem(C& c) : C(c) { }
```

```
        BaseMem(C const& c) : C(c) { }
    };
    template <typename FO1, typename FO2>
    class Composer : private BaseMem<FO1,1>,
                     private BaseMem<FO2,2> {
      public:
        // [illegible]:
        enum { NumParams = FO1::NumParams };
        typedef typename FO2::ReturnT ReturnT;
        typedef typename FO1::Param1T Param1T;
        // [illegible]: [illegible]
        Composer(FO1 f1, FO2 f2)
          : BaseMem<FO1,1>(f1), BaseMem<FO2,2>(f2) {
        }
        //“[illegible]”: [illegible]
      ReturnT operator() (typename ForwardParamT<Param1T>::Type v) {
            return BaseMem<FO2,2>::operator()
                     (BaseMem<FO1,1>::operator()(v));
        }
    };
```

[illegible] ForwardParamT [illegible] 22.6.3 [illegible]

[illegible] Abs [illegible] Sine [illegible] Abs[illegible]Sine [illegible]

```
// functors/math2.hpp
#include <cmath>
```

```
#include <cstdlib>
class Abs {
   public:
      // [illegible]:
      enum { NumParams = 1 };
      typedef double ReturnT;
   typedef double Param1T;
   //“[illegible]”:
   double operator() (double v) const {
      return std::abs(v);
   }
};
class Sine {
   public:
      // [illegible]:
      enum { NumParams = 1 };
      typedef double ReturnT;
      typedef double Param1T;
      //“[illegible]”:
      double operator() (double a) const {
         return std::sin(a);
      }
};
```

[illegible]Abs[illegible]Sine[illegible]

```
// functors/math3.hpp
#include <cmath>
#include <cstdlib>
```

```
template <typename T>
class Abs {
  public:
    // [illegible]:
    enum { NumParams = 1 };
    typedef T ReturnT;
    typedef T Param1T;
    //“[illegible]”:
    T operator() (T v) const {
      return std::abs(v);
    }
};
template <typename T>
class Sine {
  public:
    // [illegible]:
    enum { NumParams = 1 };
    typedef T ReturnT;
  typedef T Param1T;
  //“[illegible]”:
  T operator() (T a) const {
    return std::sin(a);
  }
};
```

[illegible]

```
// functors/compose5.cpp
```

```
#include <iostream>
#include "math3.hpp"
#include "compose5.hpp"
#include "composeconv.hpp"
template<typename FO>
void print_values (FO fo)
{
    for (int i=-2; i<3; ++i) {
        std::cout << "f(" << i*0.1
            << ") = " << fo(i*0.1)
            << "\n";
    }
}
int main()
{
    // □□ sin(abs(0.5))
    std::cout  <<  compose(Abs<double>(),Sine<double>
())(0.5)
            << "\n\n";
    // □□□□□□ abs()
    print_values(Abs<double>());
    std::cout << '\n';
    // □□□□□□ sin()
    print_values(Sine<double>());
    std::cout << '\n';
    // □□□□□□ sin(abs())
    print_values(compose(Abs<double>(),Sine<double>()));
```

```
std::cout << '\n';
// [illegible] abs(sin())
print_values(compose(Sine<double>(),Abs<double>()));
std::cout << '\n';
// [illegible] sin(sin())
print_values(compose(Sine<double>(),Sine<double>
()));
}
```

22.7.3 [illegible]

[illegible]Composer[illegible]1[illegible]

[illegible]Composer[illegible]1[illegible]Composer[illegible]ParamNT[illegible]operator()[illegible]20[illegible]1[illegible]Composer[illegible]

```
template <typename FO1, typename FO2>
class Composer : private BaseMem<FO1,1>,
            private BaseMem<FO2,2> {
public:
  ...
  // [illegible]0[illegible]"[illegible]":
```

```
    ReturnT operator() () {
      return BaseMem<FO2,2>::operator()
        (BaseMem<FO1,1>::operator()());
    }
    // □□1□□□□“□□□□”:
    ReturnT operator() (typename ForwardParamT<Param1T>::Type v1) {
      return BaseMem<FO2,2>::operator()
        (BaseMem<FO1,1>::operator()(v1));
    }
    //□□2□□□□“□□□□”:
    ReturnT operator() (typename ForwardParamT<Param1T>::Type v1,
            typename ForwardParamT<Param2T>::Type v2) {
      return BaseMem<FO2,2>::operator()
        (BaseMem<FO1,1>::operator()(v1, v2));
    }
    …
  };
```

□□□□□□□□□□Param1T□Param2T□□ParamNT□ParamNT □□□□□□ [23] □□□□□□□□□□□□□□□□□□Param2T□□□FunctorParam□□□□□□□□□□□□□□□□Composer□□□□□□□□typedef□

```
  template <typename FO1, typename FO2>
```

```
class Composer : private BaseMem<FO1,1>,
                 private BaseMem<FO2,2> {
   public:
      // □□□□□□□□□
      typedef typename FO2::ReturnT ReturnT;
      // □□ Param1T, Param2T□
      // - □□□□□□□□□□□□□□□□
#define ComposeParamT(N) \
         typedef   typename   FunctorParam<FO1,   N>::Type
   Param##N##T
      ComposeParamT(1);
      ComposeParamT(2);
      …
      ComposeParamT(20);
#undef ComposeParamT
   …
};
```

□□□□□□□□□ Composer □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□const□non-const□□□□□□□□□□□

```
template <typename FO1, typename FO2>
class Composer : private BaseMem<FO1,1>,
                 private BaseMem<FO2,2> {
public:
   …
   // □□□□:
   Composer(FO1 const& f1, FO2 const& f2)
      : BaseMem<FO1,1>(f1), BaseMem<FO2,2>(f2) {
```

```
    }
    Composer(FO1 const& f1, FO2& f2)
       : BaseMem<FO1,1>(f1), BaseMem<FO2,2>(f2) {
    }
    Composer(FO1& f1, FO2 const& f2)
       : BaseMem<FO1,1>(f1), BaseMem<FO2,2>(f2) {
    }
    Composer(FO1& f1, FO2& f2)
       : BaseMem<FO1,1>(f1), BaseMem<FO2,2>(f2) {
    }
    …
};
```

[illegible]

```
// functors/compose6.cpp
#include <iostream>
#include "funcptr.hpp"
#include "compose6.hpp"
#include "composeconv.hpp"
double add(double a, double b)
{
    return a+b;
}
double twice(double a)
{
    return 2*a;
}
int main()
```

```
{
   std::cout << "compute (20+7)*2: "
         << compose(func_ptr(add),func_ptr(twice))(20,7)
         << '\n';
}
```

[illegible] Composer [illegible]func_ptr[illegible]

22.8 [illegible]

[illegible]Min[illegible]

```
// functors/min.hpp
template <typename T>
class Min {
   public:
      typedef T ReturnT;
      typedef T Param1T;
      typedef T Param2T;
      enum { NumParams = 2 };
      ReturnT operator() (Param1T a, Param2T b) {
         return a<b ? a: b ;
      }
};
```

[illegible]Clamp[illegible]Min[illegible]

[illegible]

```
// functors/clamp.hpp
template <typename T, T max_result>
class Clamp : private Min<T> {
  public:
    typedef T ReturnT;
    typedef T Param1T;
    enum { NumParams = 1 };
    ReturnT operator() (Param1T a) {
      return Min<T>::operator() (a, max_result);
    }
};
```

[illegible]

22.8.1 [illegible]

[illegible]binder[illegible]3[illegible]3[illegible]3[illegible]

[illegible]3[illegible]

```
template<typename F, int P, int V>
class BindIntStatically;
  // F [illegible]
  // P [illegible]
  // V [illegible]
```

3 [illegible]

[illegible]3[illegible]switch[illegible]switch[illegible]switch[illegible]

```
...
switch (this->param_num) {
   case 1:
      return F::operator()(v, p1, p2);
   case 2:
      return F::operator()(p1, v, p2);
   case 3:
      return F::operator()(p1, p2, v);
   default:
      return F::operator()(p1, p2);       // [illegible]?
}
```

[illegible]

[illegible]binder[illegible]binder[illegible]

```
// functors/boundval.hpp
#include "typeop.hpp"
template <typename T>
class BoundVal {
   private:
      T value;
   public:
      typedef T ValueT;
```

```
        BoundVal(T v) : value(v) {
        }
        typename TypeOp<T>::RefT get() {
            return value;
        }
    };
    template <typename T, T Val>
    class StaticBoundVal {
      public:
        typedef T ValueT;
        T get() {
            return Val;
        }
    };
```

[illegible] 16.2 [illegible]

[illegible]Binder[illegible]

```
    // functors/binder1.hpp
    template <typename FO, int P, typename V>
    class Binder : private FO, private V {
      public:
        // [illegible]:
        Binder(FO& f): FO(f) {}
        Binder(FO& f, V& v): FO(f), V(v) {}
        Binder(FO& f, V const& v): FO(f), V(v) {}
        Binder(FO const& f): FO(f) {}
        Binder(FO const& f, V& v): FO(f), V(v) {}
        Binder(FO const& f, V const& v): FO(f), V(v) {}
```

```
    template<class T>
      Binder(FO& f, T& v): FO(f), V(BoundVal<T>(v)) {}
    template<class T>
      Binder(FO& f, T const& v): FO(f), V(BoundVal<T const>(v)) {}
    ...
};
```

[illegible] v[illegible]BoundVal[illegible]

22.8.2 [illegible]

[illegible]Composer[illegible]Binder[illegible]ParamNT[illegible]Binder[illegible]ParamNT[illegible] ParamNT[illegible]

[illegible]

```
// functors/binderparams.hpp
#include "ifthenelse.hpp"
template<typename F, int P>
class BinderParams {
  public:
    // [illegible]1[illegible]:
    enum { NumParams = F::NumParams-1 };
#define ComposeParamT(N)                                    \
    typedef typename IfThenElse<(N<P),
FunctorParam<F, N>, \
```

```
                    FunctorParam<F, N+1>          \
                      >::ResultT::Type                  \
            Param##N##T
        ComposeParamT(1);
        ComposeParamT(2);
        ComposeParamT(3);
        ...
    #undef ComposeParamT
    };
    □Binder□□□□□□□□□□□□□□□□□□□□
    // functors/binder2.hpp
    template <typename FO, int P, typename V>
    class Binder : private FO, private V {
      public:
        // □□□□□□□□□□□□□□□□□□□□□□□□□
        enum { NumParams = FO::NumParams-1 };
        // □□□□□□□□□□
        typedef typename FO::ReturnT ReturnT;
        // □□□□
        typedef BinderParams<FO, P> Params;
    #define
ComposeParamT(N)                                                    \
            typedef
        typename                                                      \
                    ForwardParamT<typename
                Params::Param##N##T>::Type \
          Param##N##T
```

```
        ComposeParamT(1);
        ComposeParamT(2);
        ComposeParamT(3);…
    #undef ComposeParamT
      …
    };
```

[illegible]ForwardParamT[illegible]

22.8.3 [illegible]

[illegible]Binder[illegible]Composer[illegible]3[illegible]

• [illegible]

• [illegible]

• [illegible]

[illegible]3[illegible]P[illegible]

[illegible]——[illegible]3[illegible]3[illegible]from[illegible]ArgSelect[illegible]4[illegible]

```
// functors/binder3.hpp
template <typename FO, int P, typename V>
class Binder : private FO, private V {
  public:
    …
```

```
    ReturnT operator() (Param1T v1, Param2T v2, Param3T v3) {
        return FO::operator() (ArgSelect<1>::from(v1,v1,V::get()),
                        ArgSelect<2>::from(v1,v2,V::get()),
                        ArgSelect<3>::from(v2,v3,V::get()),
                        ArgSelect<4>::from(v3,v3,V::get()));
    }
    ...
};
```

□□□□□FO::operator()□□1□□□□□□□□□operator()□□□Binder□operator□□□1□□□□□□□□□FO::operator()□□□□□□□□□□□□□□operator()□□□Binder□operator□□1□□□□□□□□□□□□FO::operator()□□□□□□□□□□A□□□□□□Binder::operator()□□□□□□□□□□□1□2□3□□□□A□P□□0□FO::operator()□A□□□□□Binder::operator()□□□□□□A□P□□0□□□FO::operator()□A□□□□□□□□□□□□A□P□□0□□□FO::operator()□A□□□□□Binder::operator()□A-1□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□3□□□□□□□□□□□□□□

```
// functors/signselect.hpp
#include "ifthenelse.hpp"
template <int S, typename NegT, typename ZeroT, typename PosT>
struct SignSelectT {
    typedef typename
        IfThenElse<(S<0),
```

```
            NegT,
            typename IfThenElse<(S>0),
                                PosT,
                                ZeroT
                            >::ResultT
            >::ResultT
        ResultT;
};
```

□□□□□□□□□□□□□□ArgSelect□□□□□□

```
// functors/binder4.hpp
template <typename FO, int P, typename V>
class Binder : private FO, private V {
    …
    private:
        template<int A>
        class ArgSelect {
            public:
                // □□□□□□□□□□□□□:
                typedef typename TypeOp<
                typename
        IfThenElse<(A<=Params::NumParams),
                                FunctorParam<Params, A>,
                                FunctorParam<Params, A-1>
                            >::ResultT::Type>::RefT
            NoSkipT;
// □□□□□□□□□□□□□:
typedef typename TypeOp<
```

```
                typename IfThenElse<(A>1),
                               FunctorParam<Params, A-1>,
                               FunctorParam<Params, A>
                             >::ResultT::Type>::RefT
            SkipT;
    // □□□□□□□:
    typedef  typename  TypeOp<typename  V::ValueT>::RefT
BindT;
    //□□□3□□□□□□□□□3□□□
    class NoSkip {
      public:
        static NoSkipT select (SkipT prev_arg, NoSkipT arg,
                               BindT bound_val) {
            return arg;
        }
    };
    class Skip {
      public:
        static SkipT select (SkipT prev_arg, NoSkipT arg,
                    BindT bound_val) {
            return prev_arg;
        }
    };
    class Bind {
      public:
        static BindT select (SkipT prev_arg, NoSkipT arg,
                    BindT bound_val) {
```

```
            return bound_val;
        }
    };
    //[illegible]:
    typedef typename SignSelectT<A-P, NoSkipT,
                       BindT, SkipT>::ResultT
            ReturnT;
    typedef typename SignSelectT<A-P, NoSkip,
                       Bind, Skip>::ResultT
            SelectedT;
    static ReturnT from (SkipT prev_arg, NoSkipT arg,
                    BindT bound_val) {
            return SelectedT::select (prev_arg, arg, bound_val);
        }
      };
    };
```

[illegible]from[illegible]Skip[illegible]NoSkipT[illegible]1[illegible]FO::operator()[illegible] 1 [illegible] v1 [illegible] v4[illegible]TypeOp<>::RefT[illegible] & [illegible]“[illegible]”[illegible]TypeOp<>::Ref[illegible] from[illegible]NoSkip[illegible]Skip [illegible] Bind[illegible]select[illegible] select [illegible]“[illegible]”[illegible]“[illegible]”[illegible]Binder[illegible]

□□□□□□□□□□□□□□□□□□□□□□Binder□□□□□□□□□□□□□□□□

```
// functors/binder5.hpp
#include "ifthenelse.hpp"
#include "boundval.hpp"
#include "forwardparam.hpp"
#include "functorparam.hpp"
#include "binderparams.hpp"
#include "signselect.hpp"
template <typename FO, int P, typename V>
class Binder : private FO, private V {
  public:
    // □□□□□□□□□□□□□□□□□□□□□□:
    enum { NumParams = FO::NumParams-1 };
    // □□□□□□□□□□□□:
    typedef typename FO::ReturnT ReturnT;
    // □□□□□:
    typedef BinderParams<FO, P> Params;
#define
ComposeParamT(N)                                            \
            typedef
    typename                                                \
              ForwardParamT<typename
      Params::Param##N##T>::Type \
      Param##N##T
  ComposeParamT(1);
  ComposeParamT(2);
  ComposeParamT(3);
```

```
    ...
#undef ComposeParamT
    // □□□□:
    Binder(FO& f): FO(f) {}
    Binder(FO& f, V& v): FO(f), V(v) {}
    Binder(FO& f, V const& v): FO(f), V(v) {}
    Binder(FO const& f): FO(f) {}
    Binder(FO const& f, V& v): FO(f), V(v) {}
    Binder(FO const& f, V const& v): FO(f), V(v) {}
    template<class T>
        Binder(FO& f, T& v): FO(f), V(BoundVal<T>(v)) {}
    template<class T>
        Binder(FO& f, T const& v): FO(f), V(BoundVal<T
const>(v)) {}
    //“□□□□”:
    ReturnT operator() () {
            return FO::operator()(V::get());
    }
    ReturnT operator() (Param1T v1) {
            return                                  FO::operator()
    (ArgSelect<1>::from(v1,v1,V::get()),
                        ArgSelect<2>::from(v1,v1,V::get()));
    }
    ReturnT operator() (Param1T v1, Param2T v2) {
            return                                  FO::operator()
    (ArgSelect<1>::from(v1,v1,V::get()),
                      ArgSelect<2>::from(v1,v2,V::get()),
```

```
                        ArgSelect<3>::from(v2,v2,V::get()));
        }
        ReturnT operator() (Param1T v1, Param2T v2, Param3T
v3) {
            return                                      FO::operator()
        (ArgSelect<1>::from(v1,v1,V::get()),
                        ArgSelect<2>::from(v1,v2,V::get()),
                        ArgSelect<3>::from(v2,v3,V::get()),
                        ArgSelect<4>::from(v3,v3,V::get()));
        }
        ...
    private:
        template<int A>
        class ArgSelect {
          public:
            // □□□□□□□□□□:
    typedef typename TypeOp<
                typename
            IfThenElse<(A<=Params::NumParams),
                          FunctorParam<Params, A>,
                          FunctorParam<Params, A-1>
                        >::ResultT::Type>::RefT
            NoSkipT;
    // □□□□□□□□□□:
    typedef typename TypeOp<
                typename IfThenElse<(A>1),
                          FunctorParam<Params, A-1>,
```

```
                              FunctorParam<Params, A>
                            >::ResultT::Type>::RefT
            SkipT;
      // □□□□□□□:
      typedef  typename  TypeOp<typename  V::ValueT>::RefT
BindT;
      // □□□□□□□□□□□3□□□□□:
      class NoSkip {
        public:
          static NoSkipT select (SkipT prev_arg, NoSkipT arg,
                                BindT bound_val) {
            return arg;
          }
      };
      class Skip {
        public:
          static SkipT select (SkipT prev_arg, NoSkipT arg,
                                BindT bound_val) {
            return prev_arg;
          }
      };
      class Bind {
        public:
          static BindT select (SkipT prev_arg, NoSkipT arg,
                                BindT bound_val) {
            return bound_val;
          }
```

```
};
// [illegible]
typedef typename SignSelectT<A-P, NoSkipT,
                        BindT, SkipT>::ResultT
        ReturnT;
typedef typename SignSelectT<A-P, NoSkip,
                        Bind, Skip>::ResultT
        SelectedT;
        static ReturnT from (SkipT prev_arg, NoSkipT arg,
                        BindT bound_val) {
        return SelectedT::select (prev_arg, arg, bound_val);
      }
   };
};
```

22.8.4 [illegible]

[illegible]

```
// functors/bindconv.hpp
#include "forwardparam.hpp"
#include "functorparam.hpp"
template <int P,          // [illegible]
        typename FO>   // [illegible]
inline
Binder<FO,P,BoundVal<typename
FunctorParam<FO,P>::Type> >
bind (FO const& fo,
```

```
        typename ForwardParamT
                <typename FunctorParam<FO,P>::Type>::Type
            val)
    {
      return Binder<FO,
                P,
                BoundVal<typename
            FunctorParam<FO,P>::Type>
              >(fo,
                BoundVal<typename
            FunctorParam<FO,P>::Type>(val)
              );
    }
```

□□□□1 □□□□□□□□□□□□□□□□□□bind()□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□

```
# include <string>
# include <iostream>
# include "funcptr.hpp"
# include "binders.hpp"
# include "bindconv.hpp"
bool func (std::string const& str, double d, float f)
{
  std::cout << str << ": "
        << d << (d<f? "<": ">=")
        << f << '\n';
    return d<f;
}
```

```
int main()
{
    bool result = bind<1>(func_ptr(func), "Comparing")
(1.0, 2.0);
    std::cout << "bound function returned " << result
<< '\n';
}
```

[illegible]bind[illegible]“[illegible]”[illegible]binder[illegible]3[illegible]double[illegible]2.0[illegible]float[illegible]float[illegible]double[illegible]

[illegible]bindfp()[illegible]bindfp()[illegible] bind [illegible]

```
// functors/bindfp2.hpp
// [illegible]2[illegible]
template<int PNum, typename RT, typename P1,
typename P2>
inline
Binder<FunctionPtr<RT,P1,P2>,
    PNum,
    BoundVal<typename
FunctorParam<FunctionPtr<RT,P1,P2>,
                PNum
                >::Type
        >
```

```
    >
bindfp (RT (*fp)(P1,P2),
    typename ForwardParamT
          <typename
      FunctorParam<FunctionPtr<RT,P1,P2>,
                  PNum
                  >::Type
          >::Type val)
{
  return Binder<FunctionPtr<RT,P1,P2>,
          PNum,
          BoundVal
            <typename
        FunctorParam<FunctionPtr<RT,P1,P2>,
                  PNum
                  >::Type
          >
        >(func_ptr(fp),
          BoundVal<typename FunctorParam
              <FunctionPtr<RT,P1,P2>,
                PNum
              >::Type
            >(val)
          );
}
```

22.9 [illegible]

[illegible] 3 [illegible]

[illegible]

```
// functors/functorops.cpp
#include <iostream>
#include <string>
#include <typeinfo>
#include "functorops.hpp"
bool compare (std::string debugstr, double v1, float v2)
{
    if (debugstr != "") {
       std::cout << debugstr << ": " << v1
                         << (v1<v2? '<' : '>')
                         << v2 << '\n';
    }
    return v1<v2;
}
void print_name_value (std::string name, double value)
{
    std::cout << name << ": " << value << '\n';
}
double sub (double a, double b)
{
    return a-b;
}
double twice (double a)
```

```
{
    return 2*a;
}
int main()
{
    using std::cout;
    // □□□□□□□:
    cout << "Composition result: "
        << compose(func_ptr(sub), func_ptr(twice))(3.0, 7.0)
        << '\n';
    // □□□□□□□:
    cout << "Binding result: "
        << bindfp<1>(compare, "main()->compare()")(1.02,
1.03)
        << '\n';
    cout << "Binding output: ";
    bindfp<1>(print_name_value,
            "the ultimate answer to life")(42);
    // □□□□□□□□□□□:
    cout << "Mixing composition and binding (bind<1>): "
        <<                                                bind<1>
(compose(func_ptr(sub),func_ptr(twice)),
            7.0)(3.0)
        << '\n';
    cout << "Mixing composition and binding (bind<2>): "
        <<                                                bind<2>
(compose(func_ptr(sub),func_ptr(twice)),
```

```
        7.0)(3.0)
      << '\n';
}
```

[illegible]

```
Composition result: -8
Binding result:      main()->compare(): 1.02<1.03
1
Binding output:      the ultimate answer to life: 42
Mixing composition and binding (bind<1>): 8
Mixing composition and binding (bind<2>): -8
```

[illegible]

[illegible]22.6.1[illegible]3[illegible]C++[illegible]func_ptr[illegible]“[illegible]”[illegible]

[illegible] functorops.hpp [illegible]

```
// functors/functorops.hpp
#ifndef FUNCTOROPS_HPP
#define FUNCTOROPS_HPP
// [illegible] func_ptr(), FunctionPtr,[illegible]FunctionPtrT
#include "funcptr.hpp"
// [illegible] Composer<>
#include "compose6.hpp"
// [illegible] compose()
```

```
#include "composeconv.hpp"
// □□ Binder<>
// -□□□□ BoundVal<>□StaticBoundVal<>□boundval.hpp
// -□□□□ ForwardParamT<>□forwardparam.hpp
// -□□□□FunctorParam<>□functorparam.hpp
// -□□□□ BinderParams<>□binderparams.hpp
// -□□□□ SignSelectT<>□signselect.hpp
#include "binder5.hpp"
// □□□□□□ bind() □ bindfp()
#include "bindconv.hpp"
#include "bindfp1.hpp"
#include "bindfp2.hpp"
#include "bindfp3.hpp"
#endif // FUNCTOROPS_HPP
```

22.10 □□□□

C++□□□□STL□□predicates□□□□□□□□□□□□□predicate□□□□□□□□Boolean□□□□□□□□□□□□□□Boolean□□□□bool□□□□□□□□□□□□□□□□□predicte□□□□□□□□□□pure functor□□□□□□□□□□□□□□□□□□□□□□□□□[JosuttisStdLib]□8.1.4□□□□

□□□□□□□C++□□□[JosuttisStdLib]□8.2□8.3□□□□□□□□□C++□□□“this and that”□□□□□□□C++□□□□□Boost□□□□□□□□□□□□□□□□□□C++□□□□□□□□

[1]. [illegible]int[illegible]built-in type[illegible]fundamental type ([illegible])[illegible]

[2]. [illegible]double[illegible]1[illegible]1[illegible]SizeOverOne[illegible]

[3]. [illegible]enum_check(int)[illegible]enum_check(unsigned int)[illegible]enum_check(singed int)[illegible]enum_check(int)[illegible]int[illegible]int[illegible]

[4]. [illegible]

[5]. [illegible]policy[illegible]

[6]. [illegible]Resource Acquisition Is Initialization[illegible]

[7]. [illegible]policy[illegible]

[8]. [illegible]C++[illegible]

[9]. [illegible]auto_ptr[illegible]C++[illegible][JosuttisAutoPtr][illegible]

[10]. [illegible]duo[illegible]“[illegible]”[illegible]2[illegible]2[illegible]trio[illegible]quartet[illegible]“[illegible]”[illegible]“[illegible]”[illegible]3[illegible]4[illegible]

[11]. [illegible]

[12]. [illegible]C++[illegible]C++[illegible]

[13]. [illegible]C++[illegible]13.3[illegible]

[14]. [illegible]

[15]. [illegible]singed[illegible]unsigned int[illegible]

[16]. [illegible]C++[illegible]

[17]. [illegible]decay[illegible]MyType::print[illegible]decay[illegible]&MyType::print[illegible]&[illegible]f[illegible]decay[illegible]&f[illegible]

[18]. [illegible]std::binary_function[illegible][JosuttisStdLib][illegible]8.2.4[illegible]

[19]. [illegible]C[illegible]C[illegible]C++[illegible]C++[illegible]

[20]. [illegible]class[illegible]15[illegible]

[21]. [illegible]

[22]. [illegible]

[23]. [illegible]SFINAE[illegible]8.3.1[illegible]SFINAE[illegible]

[illegible]A [illegible]

[illegible]C++[illegible]
ODR[illegible]One-Definition Rule[illegible]

[illegible]ODR[illegible]ODR[illegible] ODR [illegible] ODR [illegible]

A.1 [illegible]

[illegible]C++[illegible]ODR[illegible] ODR [illegible]#if[illegible]
#ifdef [illegible]#include[illegible]

[illegible]ODR[illegible]

```
//[illegible]header.hpp:
#ifdef DO_DEBUG
   #define debug(x) std::cout << x << '\n'
#else
   #define debug(x)
```

```
#endif
void debug_init();
//[illegible]myprog.cpp:
#include ”header.hpp”
int main()
{
   debug_init();
   debug(”main()”);
}
```

[illegible]

```
//[illegible]myprog.cpp
void debug_init();
int main()
{
   debug_init();
}
```

[illegible] debug_init()[illegible] exported [illegible]ADL [illegible]

[illegible]

A.2 [illegible]

[illegible] ODR [illegible] [1] [illegible]

[illegible]“[illegible]C++[illegible]”[illegible]

[illegible]

• [illegible] [illegible]“[illegible]”[illegible]

• [illegible] [illegible]

• [illegible] [illegible]

• [illegible] [illegible]

• [illegible] [illegible]extern[illegible]

• [illegible] [illegible]“[illegible]”[illegible]

• **typedefs[illegible]using-declarations[illegible]using-directive[illegible]** [illegible]typedef[illegible]union[illegible]

• [illegible] [illegible]

A.3 [illegible]

[illegible]

A.3.1 [illegible]

[illegible]

• [illegible]

• [illegible]static[illegible]

•

•

export

• export

C++ [2]

```
//1
int counter;
//2
int counter;        //ODR
```

static C++

```
//1:
static int counter = 2;          //
namespace {
   void unique()                 //{
   }
}
//2
static int counter = 0;          //
namespace {
   void unique()                 //
   {
      ++counter;
   }
}
```

```
int main()
{
   unique();
}
```

[illegible]one-per-program[illegible]“[illegible]”[illegible]new[illegible]delete[illegible]

[illegible]1[illegible]sizeof[illegible]2[illegible]1[illegible]typeid[illegible]5.6[illegible]typeid[illegible]

```
#include <typeinfo>
class Decider {
#if defined(DYNAMIC)
   virtual ~Decider() {
   }
#endif
};
extern Decider d;
int main()
{
   const char* name = typeid(d).name();
   return (int)sizeof(d);
```

}

[illegible]DYNAMIC[illegible]d[illegible]sizeof(d)[illegible]d[illegible]typeid(d)[illegible]d[illegible]d[illegible]typeid[illegible]d[illegible]

[illegible]C++[illegible]C++[illegible]

A.3.2 [illegible]

[illegible]C++[illegible]

```
inline void f() { }
inline void f() { }          //[illegible]
```

[illegible]guards[illegible]

```
//[illegible]guard_demo.hpp:
#ifndef GUARD_DEMO_HPP
#define GUARD_DEMO_HPP
...
#endif                  //GUARD_DEMO_HPP
```

[illegible]2 [illegible]#include[illegible]

ODR [illegible]class [illegible]non-exported[illegible]

[illegible]class[illegible]struct[illegible]union[illegible]X[illegible]X[illegible]

• [illegible]X[illegible]new[illegible]X[illegible]X[illegible]

• [illegible]X[illegible]

• [illegible]X[illegible]sizeof[illegible]typeof[illegible]

• [illegible]X[illegible]

• [illegible]X[illegible]X[illegible]X[illegible]X[illegible]void*[illegible]cast[illegible]static_cast[illegible]dynamic_cast[illegible]

• [illegible]X[illegible]

• [illegible] X [illegible]

[illegible]X[illegible]X[illegible]point of instantiation[illegible]POI[illegible]10.3.2[illegible]

[illegible]class[illegible]

```
inline int not_so_fast();
int main()
{
   not_so_fast();
}
inline int not_so_fast(){
}
```

[illegible]C++[illegible]

[illegible]POI[illegible]POI[illegible]exported[illegible]

[illegible]ODR[illegible]C++[illegible]C++[illegible]non-exported[illegible]

[illegible]

A.3.3 [illegible]

[illegible]C++[illegible]C++[illegible]“[illegible]”[illegible] [3] [illegible]

[illegible]

[illegible]

```
//[illegible]1[illegible]
static int counter = 0;
inline void increase_counter()
{
   ++counter;
}
int main(){
}
//[illegible]2[illegible]
static int counter = 0;
inline void increase_counter()
{
   ++counter;
```

```
}
```

[illegible] increase_counter()[illegible]counter[illegible]counter[illegible]static[illegible]

[illegible]#include [illegible] [4] [illegible]

[illegible]

```
//[illegible]1[illegible]
void unused(int = 3);
int main()
{
}
//[illegible]2[illegible]
void unused(int = 4);
```

[illegible]token stream[illegible]C++[illegible]

```
//[illegible]1[illegible]
class X {
  public:
    X(int);
    X(int, int);
};
X::X(int = 0)
```

```
{
}
class D : public X {
};
D d2;            //D()□□X(int).
//□□□□2□
class X {
  public:
    X(int);
    X(int, int);
};
X::X(int = 0, int = 0)
{
}
class D : public X {      //D()□□X(int, int);
};                                    //D()□□□□□□□ODR.
```

□□□□□□□□□□□□□□□□□□□□□□□D□□□□□□□□□□□□□□□□□□□□□□□□□□□X□□□□□□□□□□□□□□□X□□

□□“□□□□□□□□□□□□”□□

```
//□□header.hpp:
#ifndef HEADER_HPP
#define HEADER_HPP
int const length = 10;
```

```
class MiniBuffer {
   char buf[length];
   ...
};
#endif;          //HEADER_HPP
```

[illegible]length[illegible]const[illegible]static[illegible]ODR[illegible]

[illegible]POI[illegible]exported[illegible]ODR[illegible]

```
//[illegible]header.hpp
#ifndef HEADER_HPP
#define HEADER_HPP
enum Color { red, green, blue };
   //Color[illegible]
export template<typename T> void highlight(T);
void init();
#endif          //HEADER_HPP
//[illegible]tmpl_def.cpp:
#include ”header.hpp”
export template<typename T>
void highlight(T x)
```

```
{
   paint(x);            //(1)□□□□□□□□□□□□ADL
}
//□□init.cpp:
#include ”header.hpp”
namespace {   //□□□□□□□□
   void paint(Color c)         //(2)
   {
      ...
   }
}
void init()
{
   highlight(blue);       //(1)□□ADL□□□□□(2)□
}
//□□main.cpp
#include ”header.hpp”
namespace {         //□□□□□□□□
   void paint(Color c)         //(3)
   {
      ...
   }
}
int main()
{
   init();
   highlight(red);               //□1□□□ADL□□□□□□□□3□
```

}

[illegible]“[illegible]”[illegible]paint()[illegible]exported[illegible]highlight[illegible]paint()[illegible]T[illegible]POI[illegible]paint()[illegible]highlight<Color>[illegible]paint[illegible]ODR[illegible]

---

[1]. [illegible]C[illegible]C++[illegible]

[2]. [illegible]C[illegible]C[illegible]

[3]. gcc[illegible]1[illegible]Rogue[illegible]

[4]. [illegible]

[illegible]B [illegible]

[illegible]

```
void display_num(int);           //(1)
void display_num(double);      //(2)
int main()
{
   display_num(399);             //[illegible]1[illegible]
   display_num(3.99);          //[illegible]2[illegible]
}
```

[illegible]display_num()[illegible]C++[illegible]display_num()[illegible]int[illegible]1[illegible]double[illegible]

[illegible]C++[illegible]“[illegible]”[illegible]

[illegible]

B.1

B.2

```
void combine(int, double);
void combine(long, int);
int main()
{
   combine(1,2);            //
}
```

□□□□□□□combine()□□□□□□□□□□1□□□□□□□□□□□□1□□□□□
□□int□□□1□□□□2□□□□□□□□□□□□2□□□□□□□□□□□□□□□□□□□int
□long□□□□□int□double□□□□□□□□□□2□□□□□□□□C++□□□□□
□□□□□□□□□□□□□□□□□□□□□□

□□□
□□□□□□□□□□□□□□□□□□□□

• □□
□const□□volatile□□□□□

• □□□□□□□□□□□□□□□decay□□□□□□□□□□□□□□□□□const□
□□□□□int**□□□□□□□int const* const*□□□□

• □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□bool□char□
short□□□□□□□□□□□□□□□□□□int□unsigned int□long□□
unsigned long□□□□□□float□double□□□□□

• □□□□□□□□□□□□□□□□□□□□□□□□□□□□int□float□□□□□□□
□□□□□□□□□□□□□□□□□□

• □□□□□□□□□□□□□□□□□□□□□□□□□

• □□□□□□□□□□□□□□□□□□□□□□□□POD□plain old data□□□□
□□□□□□□□□□

□□□□□□□□□□□□□□

```
int f1(int);              //(1)
int f1(double);           //(2)
f1(4);                    //□□□1□□□□□,
                       //         □□2□□□□□□□
int f2(int)   ;           //□3□
int f2(char);             //(4)
f2(true);                 //□□□3□□□□□□□□□true□bool□
                       //         □□4□□□□□□□□□
```

```
class X {
   public:
      X(int);
};
int f3(X);                //(5)
int f3(...)               //(6)
f3(7);                    //□□□5□□□□□□□□□□□□,
                         //                 □(6)□□□□□□□□□
```

□□□
□□□□□□□□

```
template <typename T>
class MyString {
   public:
      MyString(T const*);         //□□□□□□□□□□
      ...
};
template <typename T>
MyString<T> truncate(MyString<T> const&, int);
int main()
{
   MyString<char> str1, str2;
   str1 = truncate<char>(”Hello World”, 5);   //□□
   str2 = truncate(”Hello World”,5);            //□□
}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□str2□□□□□
□□□□□□□□□□□truncate()□□□□□□□□□□□□□□□

[illegible]

B.2.1 [illegible]

[illegible]*this[illegible]MyClass[illegible]MyClass&[illegible]non-const[illegible]MyClass const&[illegible]const[illegible] [1] [illegible]this[illegible]this[illegible] *this [illegible]reference[illegible]this[illegible]C++[illegible]this[illegible]

[illegible]this[illegible] *this [illegible]

```
#include<stddef.h>
class BadString {
  public:
    BadString(char const*);
    ...
    //[illegible]
    char& operator[] (size_t);                    //(1)
    char const& operator[] (size_t) const;
    //[illegible]null[illegible]
    operator char*();                               //[illegible]2[illegible]
    operator char const*();
    ...
};
```

```
int main()
{
    BadString str(“correkt”);
    str[5] = ‘c’;          //[illegible]
}
```

[illegible]str[5][illegible]1[illegible]5[illegible]int[illegible]size_t[illegible]std::size_t[illegible]unsigned int[illegible]unsigned long[illegible] int [illegible]1[illegible]char*[illegible]str[illegible]str[illegible]this[illegible]char*[illegible]ptrdiff_t[illegible]ptrdiff_t[illegible]int[illegible]5[illegible]str[illegible]*this[illegible]1[illegible] [2] [illegible][][illegible]ptrdiff_t[illegible]char*[illegible]

[illegible]*this[illegible]

B.2.2 [illegible]

[illegible]int[illegible]3[illegible]int[illegible]int&[illegible]int const&[illegible]

```
void report(int&);              //(1)
void report(int const&)[illegible]     //(2)
int main()
{
```

```
    for (int k = 0; k<10; ++k) {
        report(k);                  //□□(1)
    }
    report(42);                     //□□(2)
}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ const □□□□□□□□□□□□
□□□□□□□□□□const□□□

□□□□□□□□□□□□□□□□□□□□□□

```
class Wonder {
    public:
        void tick();                //(1)
        void tick() const;          //(2)
        void tack() const;          //(3)
};
void run(Wonder& device)
{
        device.tick();              //□□□1□
        device.tack();              //□□□3□,□□□□□□□non-const
                            //          □□□Wonder::tack()
}
```

□□□
□□□□□□□□□□□

```
void report(int);                   //(1)
void report(int&);                  //(2)
void report(int const&);            //(3)
int main()
{
```

```
    for(int k = 0; k<10; ++k) {
        report(k);              //[illegible]1[illegible]2[illegible]
    }
    report(42);                 //[illegible]1[illegible]3[illegible]
}
```

[illegible]

- [illegible]T[illegible]T[illegible]T const&[illegible]
- [illegible]T[illegible]T[illegible]T&[illegible]

B.3 [illegible]

[illegible]C++[illegible]——[illegible]

B.3.1 [illegible]

[illegible]

```
template <typename T> int f(T);       //(1)
void f(int);                          //(2)
int main()
{
    return f(7);      //[illegible]2[illegible]2[illegible]
}
```

[illegible]

[illegible]12.2.2[illegible]

B.3.2 [illegible]

[illegible]

```
class Base {
   public:
   operator short() const;
};
class Derived : public Base {
};
void count(int);
void process(Derived const& object)
{
   count(object);          //[illegible]
}
```

[illegible] [3] count(object)[illegible]object[illegible]int[illegible]

1.object[illegible]Derived const[illegible]Base const[illegible]

2.[illegible]1[illegible]Base const[illegible]short[illegible]

3.[illegible]short[illegible]int[illegible]

[illegible]

[illegible]A[illegible]B[illegible]A[illegible]

```
void count(short);
```

[illegible] count(object)[illegible]3[illegible]

B.3.3 [illegible]

[illegible] [4] [illegible]

• [illegible]bool[illegible]

• [illegible]void* [illegible]

• [illegible]

• [illegible]

[illegible]“[illegible]”[illegible]

[illegible]bool[illegible]

```
void check(void*);          //(1)
void check(bool);           //(2)
void rearrange(Matrix* m)
{
   check(m);                //[illegible](1)
   ...
}
```

[illegible] void*[illegible]

[illegible]

```
class Interface {
   ...
};
class CommonProcesses : public Interface {
   ...
};
class Machine : public CommonProcesses {
   ...
};
char* serialize(Interface*);              //(1)
char* serialize(CommonProcesses*);        //(2)
```

```
void dump(Machine* machine)
{
   char* buffer = serialize(machine);      //[illegible](2)
   ...
}
```

[illegible]Machine*[illegible]CommonProcesses*[illegible]Interface*[illegible]

[illegible]

[B.3.4 [illegible]]()

[illegible]

[illegible]1[illegible]operator()[illegible]22[illegible]

[illegible]2[illegible]class[illegible] [5] [illegible]

```
typedef void FuncType(double, int);
class IndirectFunctor {
   public:
      ...
      operator()(double, double);
      operator FuncType*() const;
};
```

```
void activate(IndirectFunctor const& funcObj)
{
   funcObj(3,5);             //□□□□□□
}
```

□□funcObj(3,5)□□□□□3□□□□□□□3□□□□□□□funcObj□3□5□□□□□□□□□□operator()□□□□□□□□□□□□□□IndirectFunctor&□double□double□□□□□□□□□□□□□□□□□□□□□FuncType*□double□int□□□□□□□□□□□□□□□□□□□ operator()□□operator()□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□C++□□□□□□□□□□□□□□□□□□□□□□□

B.3.5 □□□□□□□

□□□

□1□□□□□□□□□□□□□□□□□□□□□□□□□

```
int n_elements(Matrix const&);          //(1)
int n_elements(Vector const&);          //(2)
void compute()
{
   ...
   int (*funcPtr)(Vector const&) = n_elements;    //□□(2)
   ...
}
```

□□□□□□□□□□ n_element □□funcPtr□□□□□□□□□□□□□□□□□□□

[illegible]

```
#include <string>
class BigNum {
  public:
    BigNum(long n);
    BigNum(double n);
    BigNum(std::string const&);
    ...
    operator double();
    operator long();
    ...
};
void initDemo()
{
  BigNum bn1(100103);
  BigNum bn2(”7057103224.095764”);
  int in = bn1;
}
```

[illegible] C++[illegible]std::auto_ptr[illegible]

---

[1]. [illegible]volatile[illegible]MyClass volatile&[illegible]MyClass volatile&[illegible]

[2]. [illegible]size_t[illegible]unsigned int[illegible]size_t[illegible]unsigned long[illegible]ptrdiff_t[illegible]long[illegible]

[illegible]5[illegible]

[3]. [illegible]“[illegible]”[illegible]

[4]. [illegible]pointer to members[illegible]

[5]. [illegible]const[illegible]non-const[illegible]

[illegible]

[illegible] C++ [illegible]

[illegible] http://www.josuttis.com/tmplbook [illegible]

[illegible] (newsgroups) [illegible]

[illegible]

Usenet [illegible]

[illegible] C++ [illegible]

[illegible] C++ [illegible] C++ [illegible] C++ [illegible] Usenet [illegible]

- Tutorial level C++ (unmoderated) [illegible] alt.comp.lang.learn.c-c++
- General aspects of C++ (unmoderated) [illegible] comp.lang.c++
- General aspects of C++ (moderated) [illegible] comp.lang.c++.moderated

•Aspects of the C++ standard (moderated) □ comp.std.c++

□□□□□□□□ Usenet □□□□□□□□□□□□ Google Usenet archive□ http://groups.google.com

□□□□□

[AlexandrescuDesign] Andrei Alexandrescu Modern C++Design □ Generic Programming and Design Patterns Applied Addison-Wesley,Reading,MA,2001

[AusternSTL] Matthew H.Austern Generic Programming and the STL □ Using and Extending the C++Standard Template Library Addison-Wesley,Reading,MA,1999

[BCCL] Jeremy Siek The Boost Concept Check Library http://www.boost.org/libs/concept_check/concept_check.htm

[Blitz++] Todd Veldhuizen Blitz++:Object-Oriented Scientific Computing http://www.oonumerics.org/blitz

[Boost] The Boost Repository for Free,Peer-Reviewed C++Libraries http://www.boost.org

[BoostCompose] Boost Compose Library http://www.boost.org/libs/compose

[BoostSmartPtr] Smart Pointer Library http://www.boost.org/libs/smart_ptr

[BoostTypeTraits] Type Traits Library http://www.boost.org/libs/type_traits

[CargillExceptionSafety] Tom Cargill Exception Handling: A False Sense of Security http://www.awprofessional.com/meyerscddemo/demo/magazine/index.htm C++Report,November-December 1994

[CoplienCRTP] James O.Coplien Curiously Recurring Template Patterns C++ Report, February 1995

[CoreIssue115] Core Issue 115 of the C++Standard http://anubis.dkuug.dk/jtc1/sc22/wg21/docs/cwg_toc.html

[CzarneckiEiseneckerGenProg] Krzysztof Czarnecki, Ulrich W.Eisenecker Generative Programming □ Methods,Tools,and Applications Addison-Wesley,Reading,MA,2000

[DesignPatternsGoV] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides Design Patterns Elements of Reusable Object-Oriented Software Addison-Wesley,Reading,MA,1995 [EDG] Edison Design Group Compiler Front Ends for the OEM Market http://www.edg.com [EllisStroustrupARM] Margaret A.Ellis, Bjarne Stroustrup The Annotated C++ Reference Manual(ARM) Addison-Wesley,Reading,MA,1990

[JosuttisAutoPtr] Nicolai M.Josuttis auto_ptr and auto_ptr_ref http://www.josuttis.com/libbook/auto_ptr.html

[JosuttisOOP] Nicolai M.Josuttis Object-Oriented Programming in C++ John Wiley and Sons Ltd, 2002

[JosuttisStdLib] Nicolai M.Josuttis The C++Standard Library□A Tutorial and Reference Addison-Wesley, Reading, MA, 1999

[KoenigMooAcc] Andrew Koenig,Barbara E.Moo Accelerated C++ □ Practical Programming by Example Addison-Wesley,Reading,MA,2000

[LambdaLib] Jaakko Järvi,Gary Powell LL,The Lambda Library http://www.boost.org/libs/lambda/doc

[LippmanObjMod] Stanley B.Lippman Inside the C++ Object Model Addison-Wesley, Reading, MA, 1996

[MeyersCounting] Scott Meyers Counting Objects In C++ C/C++Users Journal,April 1998

[MeyersEffective] Scott Meyers Effective C++ 50 Specific Ways to Improve Your Programs and Design(2nd Edition)Addison-Wesley,Reading,MA,1998

[MeyersMoreEffective] Scott Meyers More Effective C++ 35 New Ways to Improve Your Programs and Designs Addison-Wesley,Reading,MA,1996

[MTL] Andrew Lumsdaine, Jeremy Siek MTL, The Matrix Template Library http://www.osl.iu.edu/research/mtl

[MusserWangDynaVeri] D.R.Musser, C.Wang Dynamic Verification of C++ Generic Algorithms IEEE Transactions on Software Engineering,Vol.23,No.5,May 1997

[MyersTraits] Nathan C.Myers Traits: A New and Useful Template Technique http://www.cantrip.org/traits.html

[NewMat] Robert Davies NewMat10,A Matrix Library in C++ http://www.robertnz.com/nm_intro.htm

[NewShorterOED] Leslie Brown, et al.The New Shorter Oxford English Dictionary (fourth edition) Oxford University Press,Oxford,1993

[POOMA] POOMA: A High-Performance C++ Toolkit for Parallel Scientific Computation http://www.pooma.com

[Standard98] ISO Information Technology--Programming Languages--C++ Document Number ISO/IEC 14882-1998 ISO/IEC 1998

[Standard02] ISO Information Technology--Programming Languages--C++ (as amended by the first technical corrigendum) Document Number ISO/IEC 14882-2002 ISO/IEC, expected late 2002

[StroustrupC++PL] Bjarne Stroustrup The C++ Programming Language, Special ed.Addison-Wesley, Reading, MA, 2000

[StroustrupDnE] Bjarne Stroustrup The Design and Evolution of C++ Addison-Wesley, Reading, MA, 1994

[StroustrupGlossary] Bjarne Stroustrup Bjarne Stroustrup's C++ Glossary http://www.research.att.com/~bs/glossary.html

[SutterExceptional] Herb Sutter Exceptional C++ □ 47 Engineering Puzzles, Programming Problems,and Solutions Addison-Wesley,Reading,MA,2000

[SutterMoreExceptional] Herb Sutter More Exceptional C++ □ 40 New Engineering Puzzles,Programming Problems,and Solutions Addison-Wesley,Reading,MA,2001

[UnruhPrimeOrig] Erwin Unruh Original Metaprogram for Prime Number Computation http://www.erwin-unruh.de/primorig.html

[VandevoordeSolutions] David Vandevoorde C++ Solutions Addison-Wesley, Reading, MA, 1998

[VeldhuizenMeta95] Todd Veldhuizen Using C++Template Metaprograms C++ Report, May 1995

[VeldhuizenPapers] Todd Veldhuizen Todd Veldhuizen's Papers and Articles about Generic Programming and Templates http://osl.iu.edu/ tveldhui/papers

[illegible]

[illegible]C++[illegible][StroustrupGlossary][illegible]

abstract class [illegible]

[illegible]ABC[illegible]Abstract Base Class[illegible]

ADL

argument-dependent lookup[illegible]ADL[illegible]ADL[illegible]Koenig[illegible]Koenig[illegible]ADL[illegible]

[illegible]hack

[illegible] > [illegible]vector<list<int>>[illegible]C++[illegible]hack[illegible]vector<list<int> >[illegible]

[illegible]

[illegible] < [illegible] > [illegible]

ANSI

American National Standard Institute[illegible] J16 [illegible] C++[illegible]ANSI [illegible] ISO [illegible]international standard organization[illegible]

argument [illegible]

[illegible]abs(-3)[illegible]-3[illegible]actual arguments[illegible]parameters[illegible]formal parameters[illegible]

argument-dependent lookup[illegible]ADL[illegible]

class [illegible]

[illegible]C++[illegible]class[illegible]struct[illegible]

class template [illegible]

[illegible]

class type class [illegible]

[illegible]class[illegible]struct[illegible]union[illegible]C++[illegible]

collection class [illegible]

[illegible]C++[illegible]

constant-expression

[illegible]true constant[illegible]constant expression[illegible]-[illegible]constant expression[illegible]

const member function const[illegible]

[illegible]*this[illegible]

container [illegible]

conversion operator [illegible]

[illegible]operator type()[illegible]

CRTP

Curiously Recurring Template Pattern(□□□□□□□□□)□□□□□□□□

□□□□□□□□X□□□□□□□□□□□□X□□□□□□□□□□□

Curiously Recurring Template Pattern □□□□□□□□□□ □

CRTP□

decay [1]

□□□□□□□□□□□□□□□□□□□□□□□□□□□□“hello”□□□□char const

[6]□□□□□C++□□□□□□□□□□□□□□□□char const*□□□□□□□□□□□□

□□□□□□□

declaration □□

□□□□□□□□□□□□□□□□□□□C++□□□□□□□□□□“□□”□

deduction □□

□□□

definition □□

□□□

□□□□□□□□□class□□

□□□□□□□□extern□□□□□□□□□□□□□□□□□

dependent base class □□□□□

□□

dependent name □□□□□

□□□□□□□□□□□□□□□□A□□T□□□□□□□□□□□□A<T>::x□□□□□□□□□□□

□□□

□□□□f((T*)0)□□□T□□□□□□□□□□□f□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□T□□□□□□□□□□□□□□□□□

digraph □□ [2]

□□□□□□□□□□□□□□□C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

C++□□

□::□□□□□□□□□□□□□□ <: □

dot-C file dot-C

dot-C dot-C .cpp .C .c .cc .cxx

EBCO

Empty Base Class Optimization

Empty Base Class Optimization **EBCO**

explicit instantiation directive

POI point of instantiation C++

explicit specialization

C++

expression template

friend name injection

full specialization

function object

function template

functor [3]

C++

operator()

header file □□□

□□□□#include □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□ □□□□□□□□□□□□ .hpp□.h□.H□.hh□.hxx□□□□□□□□□□□□□□□□□□□□□□ dot-C□□□□□□□□

include file □□□□□□ □□□□□

indirect call □□□□

□□□

initializer □□□□ [4]

□□□□□□□□□□□□□□□□□□□□□□□□□□□

std::complex<float> z1 = 1.0, z2(0.0,1.0);

□□□□□□□□ = 1.0 □□0.0□1.0□□

initializer list □□□□□

□□□ □□□□□□□□□□□□□□□□□□□□□□□

injected class name □□□□□□

□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□

instance □□

□C++□□ —□□□□□□□□□□□□□□□□□□□C++□□std::cout□□□std::ostream□□□□ □□ □□ □□□□

instantiation □□□

□□□ □□□□□□□□□□□□□□□□□□□□□□□□□

ISO

International Organization for Standard□□□□□□□□□□□□□□□□□□WG21□ISO□□□□□□C++□□□□□□□□

iterator □□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

linkable entity □□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□,□□□□□□□□□□□□□□□□

lvalue □□

□□□C□□rvalue□□□□□□□C□C++□□□[]□□□T()□□□□□□□□□□□□□□□□

member class template □□□□□

□□□

member function template □□□□□□

□□

member template □□□□

□□□□□□□□□□□□□□□

nondependent name □□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□

ODR

one-definition rule [illegible] C++ [illegible] 7.4 [illegible] A [illegible]

one-definition rule [illegible] ODR [illegible] one-definition rule [illegible]

overload resolution [illegible]

[illegible]

parameter [illegible]

[illegible]

parameterized class [illegible]

[illegible]

parameterized function [illegible]

[illegible] 3 [illegible]

partial specialization [illegible]

[illegible]

POD

Plain Old Date(type) [illegible] POD [illegible] C++ [illegible] C [illegible] struct [illegible] POD [illegible]

POI

Point Of Instantiation [illegible] POI [illegible]

POI[illegible]

Point Of Instantiation [illegible]POI[illegible]

policy class policy[illegible]

[illegible]policy[illegible] policy[illegible]policy[illegible]policy[illegible]policy[illegible]trait[illegible]

polymorphism [illegible]

[illegible]C++[illegible]C++[illegible]

precompiled header [illegible]

[illegible]C++[illegible]

primary template [illegible]

“[illegible]”[illegible]

qualified name [illegible]

[illegible]::[illegible]

reference counting [illegible]

[illegible]0[illegible]

rvalue [illegible]

source file [illegible]

[illegible]dot-C[illegible]

specialization [illegible]

[illegible]

template [illegible]

[illegible]

template argument [illegible]

[illegible]

template argument deduction [illegible] [illegible]

template-id

[illegible]std::list<int>[illegible]

template parameter [illegible]

[illegible]

trait template trait [illegible]

[illegible]trait[illegible]trait[illegible]policy[illegible]

translation unit [illegible]

[illegible]dot-C[illegible]#include[illegible]#if [illegible]dot-C[illegible]dot-C [illegible]

true constant [illegible]constant-expression[illegible]

tuple

C struct[illegible]

two-phase lookup [illegible]

[illegible]1[illegible]2[illegible]::)[illegible]2[illegible]2[illegible]ADL[illegible]

user-defined conversion [illegible]

[illegible]explicit[illegible]

whitespace [illegible]

[illegible]C++[illegible]

---

[1]. [illegible]decay[illegible]

[2]. [illegible]digraph[illegible]

[3]. [illegible] functor[illegible]“[illegible]”[illegible]“[illegible]”[illegible]

[4]. [illegible]